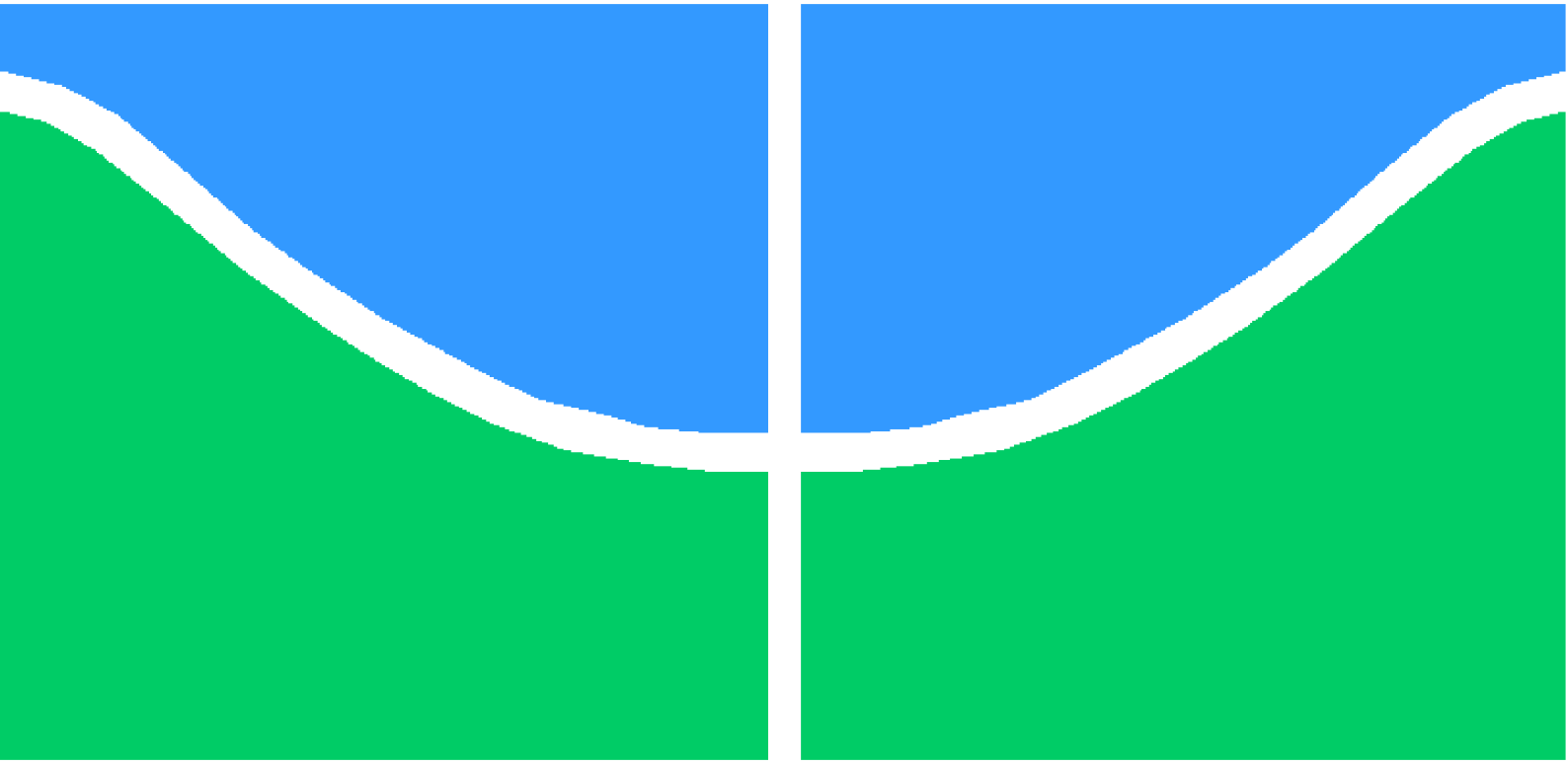


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

ANÁLISE DE IMPACTO DO USO DE TÉCNICAS DE PROGRAMAÇÃO NO DESEMPENHO DE APLICAÇÕES ANDROID™

Autor: Victor Hugo Alves de Carvalho

Orientador: Msc Fabiana Freitas Mendes

Brasília, DF

2015

Victor Hugo Alves de Carvalho

ANÁLISE DE IMPACTO DO USO DE TÉCNICAS DE PROGRAMAÇÃO NO DESEMPENHO DE APLICAÇÕES ANDROID™

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Msc Fabiana Freitas Mendes

Brasília, DF

2015

Victor Hugo Alves de Carvalho
ANÁLISE DE IMPACTO DO USO DE TÉCNICAS DE PROGRAMAÇÃO NO DESEMPENHO DE APLICAÇÕES ANDROID™/ Victor Hugo Alves de Carvalho. – Brasília, DF, 2015-
103 p. : il. (algumas color.) ; 30 cm.

Orientador: Msc Fabiana Freitas Mendes

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Android. 2. Desempenho. I. Msc Fabiana Freitas Mendes. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. ANÁLISE DE IMPACTO DO USO DE TÉCNICAS DE PROGRAMAÇÃO NO DESEMPENHO DE APLICAÇÕES ANDROID™

CDU 02:141:005.6

Victor Hugo Alves de Carvalho

ANÁLISE DE IMPACTO DO USO DE TÉCNICAS DE PROGRAMAÇÃO NO DESEMPENHO DE APLICAÇÕES ANDROID™

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 02 de julho de 2015:

Msc Fabiana Freitas Mendes
Orientador

Msc Ricardo Ajax Dias Kosloski
Convidado 1

Dr Wander Cleber Maria Pereira da Silva
Convidado 2

Brasília, DF
2015

Este trabalho é dedicado a minha família que,
mesmo nos momentos difíceis não deixaram de me apoiar.

Agradecimentos

À Deus que me deu forças e bençãos para prosseguir minha caminhada.

Aos meus pais e irmãos que nunca me abandoram e sempre acreditaram em mim.

A minha esposa Mírian que contribuiu para o meu desenvolvimento profissional.

Aos meus companheiros de trabalho que deram todo apoio durante os cinco anos que cursei a Universidade.

A minha orientadora, Fabiana Freitas Mendes, pela orientação concedida e compreensão durante a elaboração dos trabalhos.

A todos aqueles que de uma forma ou de outra contribuíram para a realização deste trabalho.

Resumo

A plataforma Android cresceu rapidamente desde o seu lançamento, não só em número de dispositivos, mas também em relação à presença nos mercados. Por causa da facilidade de se programar, da flexibilidade da plataforma e confiança do mercado, empresas de base tecnológica têm adotado esta plataforma, utilizando sistemas com Android nas mais diversas áreas. Devido à complexidade de se melhorar o desempenho de um aplicativo, este trabalho tem como objetivo comparar o uso e desuso de técnicas de programação por meio de métricas. Para tanto, foi realizada pesquisa aplicada, bibliográfica, quantitativa e experimental como metodologia de pesquisa. Foram estudas as técnicas de programação, métricas de software, ferramentas para coleta de dados, e o software para aplicação das técnicas selecionadas. Estes estudos resultaram em um experimento que mostrou o indício de que existem técnicas de programação, que o impacto do uso delas é positivo e maior do que outras técnicas de programação nos recursos CPU e memória dos dispositivos móveis que utilizam o Sistema Operacional Android.

Palavras-chaves: Android. Desempenho. Software. Métricas de Software. Ferramentas.

Abstract

The Android platform has grown rapidly since its launch, not only in number of devices, but also on their market presence. Because of the ease of programming, platform flexibility and confidence of the market, technology-based companies have adopted this platform, using systems with Android in several areas. Due to the complexity of improving the performance of an application, this work aims at comparing the use and disuse of programming techniques using metrics. Therefore, research was conducted applied, literature, quantitative and experimental as a research methodology. They were studied programming techniques, software metrics, tools for data collection, and the software for application of selected techniques. These studies resulted in an experiment that showed evidence that there are programming techniques that the impact of the use of them is positive and higher than other programming techniques in the resources CPU and memory of mobile devices using the Android operating system.

Keywords: Android. performance. Software. Software Metrics. Tools.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A necessidade de comunicação em tempo real fez com que o celular alcançasse a utilização em grande escala. Segundo os últimos dados da ANATEL (Agência Nacional de Telecomunicações) (ANATEL, 2015), em março de 2015 o Brasil fechou com 283,4 milhões de linhas de telefones móveis ativas, por isso torna-se cada vez mais importante desenvolver aplicativos voltados para esse segmento. Criar um aplicativo para dispositivos móveis requer diversos desafios e complicações como: lidar com recursos físicos limitados tais como CPU, memória, tela, dispositivos de entrada, bateria (PEREIRA; SILVA, 2009).

Inicialmente, o celular tinha apenas propósito de prover serviço de telefonia móvel, mas nos últimos anos ocorreu uma evolução nos dispositivos que passaram a oferecer variadas possibilidades para o desenvolvimento de aplicações, aumentando assim a utilidade do celular. O Android é um sistema operacional de código aberto que permite acessar qualquer funcionalidade do núcleo do telefone como chamadas, mensagens, câmera e outras para que sejam adaptadas e aprimoradas visando sempre a evolução do sistema e melhor experiência de uso pelo usuário final (PEREIRA; SILVA, 2009).

Neste documento são abordadas as técnicas de programação que, segundo mantenedor do Android, a Google, afetam o desempenho de um software desenvolvido utilizando a plataforma Android (OHA, 2009). Também será tratada a criação de métricas de qualidade, visando o estabelecimento de dados para comprovação do impacto individual das técnicas de programação com relação a um determinado recurso relacionado ao desempenho em dispositivos móveis.

Este capítulo será organizado da seguinte maneira: primeiramente será mostrado o contexto e a motivação de estudo, seguido da descrição do problema, objetivos gerais e específicos, metodologia escolhida, terminando com o cronograma deste trabalho.

1.1 Contexto e Motivação do Estudo

O Android é um sistema operacional baseado em Linux e com código aberto. Foi idealizado para aparelhos com especificações diversas (OHA, 2009) e suporta tecnologias presentes nos smartphones mais modernos atualmente como *touchscreen* (tela sensível ao toque) e GPS (*Global Positioning System* – Sistema de Posicionamento Global), tendo sido desenvolvido pela Google para smartphones e tablets.

Mesmo baseado no sistema Linux, Pereira e Silva (2009) não consideram o sistema operacional Android um Linux completo. Usado e customizado por várias empresas de eletrônicos, o sistema encontra-se em sua versão 5.0, chamada Lollipop. Diversas empresas,

dentre elas a Google, juntaram-se para formar, em novembro de 2007, a OHA (Open Handset Alliance) cujo objetivo é, no contexto do Android, acelerar e trazer inovação em aplicações e serviços.

A plataforma Android é uma solução completa para tecnologia móvel, oferecendo um pacote com diversos programas para celular, já com um sistema operacional, aplicativos e interface para o usuário.

Uma grande vantagem do sistema é o grande número de desenvolvedores que contribuem na criação de aplicativos. O problema intrínseco a esse processo é o grande número de *malwares* e aplicativos com falhas de segurança que acabam sendo desenvolvidos em massa. Por ser *open source*, pode ser sempre adaptado a fim de incorporar novas tecnologias. A plataforma está em constante evolução, já que há várias comunidades de desenvolvedores trabalhando em conjunto para construir aplicações inovadoras, buscando um melhor uso dos recursos existentes nos dispositivos móveis (PEREIRA; SILVA, 2009).

Pode-se dizer que o desempenho é a relação entre a quantidade de recursos, quantidade de processamento e tempo utilizado para executar certo algoritmo, podendo ainda sofrer influência de fatores internos e externos. Como exemplo de fatores internos pode-se citar instruções provenientes do código fonte, como apresentação de gráficos na tela ou gravação de dados. Fatores externos são variáveis não controladas pelos desenvolvedores do código, como ambiente e rede (PEREIRA; SILVA, 2009).

A busca de desempenho no software para ambientes móveis é complexa, os componentes são variáveis no tempo e espaço em termos de portabilidade, mobilidade e conectividade. Porém, existem também características comuns como a capacidade das aplicações adaptarem sua funcionalidade às condições dos recursos envolvidos nos diferentes momentos da execução (YAMIN et al., 2001).

Este trabalho, portanto, insere-se nesse contexto da avaliação do desempenho em aplicativos desenvolvidos utilizando o Sistema Operacional Android.

1.2 Descrição do Problema

Em aplicações para dispositivos móveis, o requisito não funcional de desempenho é prioridade. Durante o desenvolvimento de um aplicativo, os recursos utilizados são provenientes da máquina do desenvolvedor e dos ambientes de teste. Ao se distribuir esse aplicativo, ele será utilizado em diferentes ambientes, com baixo ou alto qualidade de recursos computacionais. Por poder ser executado em vários dispositivos e por muitos deles possuírem poucos recursos, o sistema operacional monitora o uso de recursos, podendo até encerrar a execução do aplicativo.

Programadores, em sua formação, aprendem que a aplicação de técnicas de pro-

gramação, ou seja, de boas práticas de desenvolvimento, auxiliam na redução do esforço de manutenção de software.

São sérias as consequências práticas de desenvolver software de maneira desordenada, sem utilização de processos definidos e de melhores práticas de desenvolvimento de software, por exemplo, softwares difíceis de se dar manutenção, tanto corretiva quanto evolutiva (ENGHOLM, 2010).

De acordo com (PEREIRA; SILVA, 2009), existem diversos desafios e complicações em relação a dispositivos móveis como: lidar com recursos físicos limitados tais como CPU, memória, tela, dispositivos de entrada, bateria. Devido a essa complexidade, esse trabalho avalia a utilização de técnicas em aplicativos desenvolvidos utilizando o Sistema Operacional Android, buscando encontrar indícios de qual a melhor técnica de programação em relação a um recurso limitado dos dispositivos móveis.

1.3 Objetivos Geral e Específico

Motivado pela necessidade de aplicativos cada vez mais eficientes, pelo crescente número de dispositivos móveis utilizando o sistema operacional Android e pela quantidade limitada de recursos desses dispositivos, o objetivo deste trabalho é avaliar o impacto de um conjunto de técnicas de programação no desempenho de um aplicativo desenvolvido para Android, a fim de encontrar indícios de qual a melhor técnica de programação em relação a um determinado recurso desses dispositivos, visando um melhor desempenho. Para tanto, é necessário atingir os seguintes objetivos específicos:

- Identificar os recursos que mais impactam no desempenho de uma aplicação Android do ponto de vista do usuário final e dos desenvolvedores.
- Selecionar técnicas de programação e implementá-las em uma aplicação Android, a fim de avaliar se realmente influenciam o desempenho.
- Avaliar o impacto individual de cada técnica de programação no desempenho da aplicação.
- Encontrar indícios de qual a melhor técnica de programação em relação a um determinado recurso desses dispositivos, visando um melhor desempenho.

1.4 Metodologia

De acordo com (SILVA; MENEZES, 2001), a pesquisa utilizada neste TCC pode ser classificada como:

- Pesquisa Aplicada, pois visa produzir conhecimentos para aplicação prática, voltados para o problema de desempenho em aplicações Android.
- Pesquisa Quantitativa, pois busca mostrar através de números o impacto do uso das técnicas de programação.

Em relação ao método utilizado para alcançar os objetivos específicos propostos, esta pesquisa pode ser classificada de três formas:

- Pesquisa Bibliográfica, pois busca informações com o uso de computadores e em informações publicadas em livros, revistas e artigos científicos.
- Pesquisa Experimental, pois define um programa piloto para testar o impacto do uso das técnica de programação.
- Survey, pois busca mostrar através de questionário o recurso do dispositivo móvel mais importante quando se refere ao desempenho analisando um determinado aplicativo.

1.5 Cronograma

Nas Tabelas 1 e 2, respectivamente, é mostrado o cronograma do TCC 1 e TCC 2. Cada parte está estruturada com uma tabela contendo as etapas e suas respectivas datas de execução, em que cada X corresponde a semana para realização de uma etapa. Essas etapas serão descritas contendo uma breve explicação.

Em relação a TCC 1, foi executado o cronograma apresentado na Tabela 1.

Tabela 1 – Cronograma TCC 1

MESES/SEMANAS																
Etapas	Março			Abril					Maio				Junho			
	1	**2**	**3**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Etapa 1	**x**															
Etapa 2		**x**	**x**													
Etapa 3				**x**	**x**											
Etapa 4					**x**	**x**										
Etapa 5						**x**	**x**									
Etapa 6								**x**	**x**							
Etapa 7										**x**	**x**	**x**	**x**			
Etapa 8														**x**		

Como pode ser visto na Tabela 1, o TCC1 foi dividido em oito etapas, sendo elas:

Etapa 1 – Apresentação do tema. Aqui foi apresentado e discutido tema do TCC para a professora orientadora.

Etapa 2 – Início do Trabalho de Conclusão do Curso (TCC). Nesta etapa foi elaborada a introdução, contexto e motivação de estudo, descrição do problema, objetivos (geral e específico), cronograma e metodologia.

Etapa 3 – Técnicas de Programação listadas pela Google. Nesta etapa foram selecionadas as técnicas de programação citadas pela Google.

Etapa 4 – Plano de medição. Nesta etapa foram elaborados os planos de medição.

Etapa 5 – Ferramentas para coleta de dados. Nesta etapa foram selecionadas e analisadas algumas ferramentas que poderiam ser utilizadas para coleta de dados.

Etapa 6 – Decisão sobre o software que será analisado. Aqui foram analisadas diversas aplicações Android que poderiam ser utilizadas para testar as técnicas de programação selecionadas.

Etapa 7 - Seleção de técnicas, ferramentas e software. Nesta etapa foram analisadas técnicas, ferramentas e softwares, em seguida, foi selecionada quais os que foram utilizados para a produção do TCC.

Etapa 8 – Revisão do trabalho. Nesta etapa o trabalho escrito deste TCC foi revisado.

Em relação a TCC 2, foi executado o cronograma apresentado na Tabela 2.

Tabela 2 – Cronograma TCC 2

MESES/SEMANAS																	
Etapas	**Março**				**Abril**					**Maio**				**Junho**			
	1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Etapa 1	x	x															
Etapa 2		x															
Etapa 3			x	x													
Etapa 4				x	x	x											
Etapa 5							x	x									
Etapa 6									x	x							
Etapa 7											x	x					

Como pode ser visto na Tabela 2, o TCC 2 foi dividido em oito etapas, sendo elas:

Etapa 1 – Detalhamento das métricas que não foram detalhadas no TCC1. Nesta etapa foram detalhadas as outras métricas que estavam definidas no TCC1.

Etapa 2 – Estudo sobre desenvolvimento de experimento. Nesta etapa foram estudadas as formas de se realizar um experimento na engenharia de software.

Etapa 3 – Elaboração do planejamento para realização do experimento. Após o estudo sobre o desenvolvimento de um experimento na engenharia de software, foi elaborada a parte do planejamento do experimento.

Etapa 4 – Operação do experimento. Após a elaboração do planejamento do experimento, foi executa a parte da operação.

Etapa 5 – Análise e Interpretação dos dados coletados. Após a operação do experimento, foram análisados e intepretados os dados gerados pela medição.

Etapa 6 - Elaboração da conclusão do TCC. Após a conclusão do experimento, foi elaborada a conclusão deste TCC.

Etapa 7 – Revisão do trabalho. Nesta etapa o trabalho escrito deste TCC foi revisado.

Este documento estará organizado da forma como se segue:

- **Capítulo 2**: Mostra um objetivo para o desempenho, posteriormente como ele é medido em aplicações móveis e por último a escolha dos recursos que foram analisados neste trabalho
- **Capítulo 3**: Mostra a pesquisa realizada sobre as técnicas de programação que afetam o desempenho de uma aplicação Android, em seguida são apresentadas aquelas que serão alvos de estudo nesse trabalho.
- **Capítulo 4**: Mostra o protocolo de pesquisa definido para a busca de ferramentas que será realizada em TCC 2, as ferramentas e o software que foram utilizados para realizar a coleta dos dados e a implementação da técnica de programação.
- **Capítulo 5**: Mostra a pesquisa realizada sobre as possíveis métricas de software e a escolha de qual será abordada.
- **Capítulo 6**: Detalha o experimento realizado neste trabalho e apresenta os resultados por ele alcançados.
- **Capítulo 7**: Apresenta as considerações finais deste trabalho com um resumo dos principais resultados obtidos por esta pesquisa, bem como os trabalhos futuros associados.

2 Desempenho

Este capítulo está estruturado da seguinte forma: inicialmente são apresentados alguns conceitos relacionados a desempenho, logo após a forma como mede-se o desempenho em aplicações móveis e por último a escolha dos recursos que foram analisados neste trabalho.

2.1 Introdução

A indústria de software tem investido cada vez mais recursos na busca pela qualidade de seus produtos. Segundo (PEREIRA; SILVA, 2009), a qualidade é composta pela funcionalidade, eficiência, confiabilidade, portabilidade, usabilidade e manutenibilidade. Sendo assim, o objetivo do desempenho é avaliar exclusivamente a eficiência e a confiabilidade.

- Eficiência: habilidade de garantir o desempenho requerido sob certas condições.
- Confiabilidade: habilidade de prover o nível requerido de serviço quando o software é usado sob as condições apropriadas.

Além disso, pode-se dizer que o desempenho é a relação entre a quantidade de recursos, quantidade de processamento e tempo utilizado para executar certo algoritmo, podendo ainda sofrer influência de fatores internos e externos (PEREIRA; SILVA, 2009).

2.2 Desempenho em aplicações móveis

A busca de desempenho no software para ambientes móveis é complexa, os componentes são variáveis no tempo e espaço em termos de portabilidade, mobilidade e conectividade. Porém, existem também características comuns como a capacidade das aplicações adaptarem sua funcionalidade às condições dos recursos envolvidos nos diferentes momentos da execução (YAMIN et al., 2001).

De acordo com a (GOOGLE, 2013), os principais componentes dos dispositivos móveis são: CPU (Processador), memória, disco/Armazenamento permanente, bateria e fontes de alimentação, portas de conexão, tela, teclado e periféricos (câmera, GPS, redes).

2.3 Survey

Neste trabalho foi construído um survey para levantamento de dados com o objetivo de encontrar o recurso do dispositivo móvel mais importante quando se refere ao desempenho analisando um determinado aplicativo. De acordo com (FINK, 2003), survey pode ser definido como um método para coletar informação de pessoas acerca de suas ideias, sentimentos, planos, crenças, bem como origem social, educacional e financeira. Esse tipo de validação foi escolhido baseado em outros trabalhos que também realizaram esse tipo de validação (SILVA, 2014; BRITO, 2014).

2.3.1 Construção do questionário

O instrumento utilizado no survey foram dois questionários , para medir a opinião do respondente. O levantamento de dados por amostragem, ou survey, assegura melhor representatividade e permite generalização para um população mais ampla. A pesquisa survey é classificada como um pesquisa exploratória, pois visa identificar quais conceitos são adequados para serem medidos em dada situação (FREITAS et al., 2000). No caso deste trabalho, a pesquisa survey busca encontrar os recursos do dispositivo móvel mais importantes quando se refere ao desempenho analisando um determinado aplicativo.

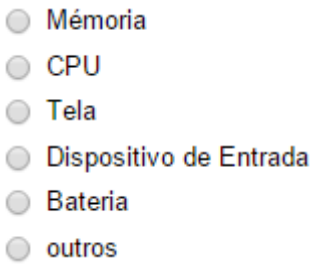
De acordo com (GUNTHER, 2003), para elaborar um questionário deve-se primeiro definir o objetivo de pesquisa em termos de conceitos a serem pesquisados e da população-alvo.

O objetivo do primeiro questionário foi encontrar os recursos do dispositivo móvel mais importantes quando se refere ao desempenho ao se usar um determinado aplicativo. Do segundo questionário foi encontrar qual o recurso do dispositivo móvel mais importante quando está se desenvolvendo um aplicativo para a plataforma Android.

Para determiner a amostragem do intrumento de pesquisa do primeiro questinário foi levada em consideração que a população-alvo é um grupo de usuários que utilizam dispositivos móveis. Para o segundo questionário foi levada em consideração que a população-alvo é uma comunidade restrita de desenvolvedores que criam aplicativos voltados para o sistema operacional Android. Dessa maneira, foi definido utilizar uma amostra não probabilística por conveniência, na qual os participantes são escolhidos por estarem disponíveis (FREITAS et al., 2000).

O questionário foi construído na ferramenta disponibilizada pela Google, com o intuito de facilitar a aplicação e a coleta de dados. O item que compôs o questionário foi o próprio objetivo. Para cada objetivo o respondente é convidado a definir qual o recurso do dispositivo móvel mais importante de acordo com as alternativas apresentadas na Figura 1 que são representados pelos recursos listados na Seção 2.2 deste capítulo.

Figura 1 – Alternativas das perguntas dos questionários



2.3.2 Coleta de dados

Os dois questionários foram transferidos para uma plataforma online, gratuita do Google Drive, no qual foi garantido o sigilo e a privacidade do respondente. O primeiro e o segundo questionários foram disponibilizados, respectivamente, por meio dos links <http://goo.gl/forms/XbB4H3fCNx> e <http://goo.gl/forms/iD6StNdAq7>.

Através da rede social Facebook, o primeiro questionário foi disponibilizado para qualquer usuário que tenha um dispositivo móvel. O instrumento foi disponibilizado durante um período de oito dias.

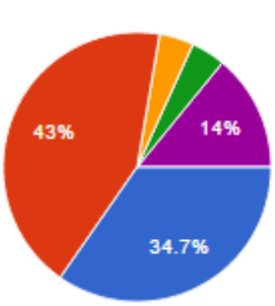
Através da rede social Facebook, o segundo questionário foi disponibilizado para uma comunidade restrita de desenvolvedores Android. O instrumento foi disponibilizado durante um período de seis dias.

2.3.3 Resultado

Durante esse prazo, o primeiro questionário foi respondido por 121 pessoas e o segundo por 41 pessoas. As respostas preenchidas pelos respondentes foram registradas pela ferramenta que disponibilizou o instrumento na internet. Os dados foram coletados a partir da planilha eletrônica disponibilizada pela ferramenta da Google. Os dados coletados podem ser encontrados nas Figuras 2 e 3.

A Figura 2 representa as respostas referentes ao primeiro questionário.

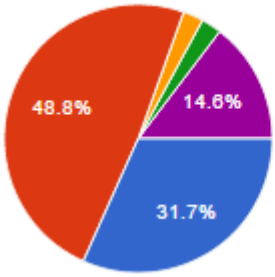
Figura 2 – Resultado da enquete disponibilizada para os usuários de dispositivos móveis



Memória	42	34.7%
CPU (Processador)	52	43%
Tela	5	4.1%
Dispositivos de Entrada	5	4.1%
Bateria	17	14%
Outros	0	0%

A Figura 3 representa as respostas referentes ao primeiro questionário.

Figura 3 – Resultado da enquete disponibilizada para desenvolvedores de aplicativos para o sistema operacional Android



Mémoria	**13**	31.7%
CPU	**20**	48.8%
Tela	**1**	2.4%
Dispositivo de Entrada	**1**	2.4%
Bateria	**6**	14.6%
outros	**0**	0%

2.4 Considerações Finais

Para este trabalho foi escolhido a Memória e CPU, tendo em vista que foram os recursos mais votados durante a enquete. A bateria também obteve um número grande de votos, mas foi descartada -neste TCC- por não se ter recursos financeiros para adquirir o que é necessário para simular testes que controlem variáveis externas como sinal do dispositivo móvel, uma vez que essa busca pelo sinal acarreta em um consumo de bateria, tendo como consequência disso a análise não somente do item em questão.

3 Técnicas de Programação

Este capítulo está estruturado da seguinte forma: inicialmente é apresentada a justificativa para a escolha do Android, depois a justificativa para a escolha da Google como referência de estudo, a justificativa da escolha do foco do trabalho, seguido das dicas de desempenho que foram selecionadas.

3.1 Escolha do Android

Com o crescente número de opções de dispositivos Android no mercado, cada vez mais as pessoas se perguntam: “qual o melhor dispositivo pra mim?”. No cenário de dispositivos celulares móveis, existem muitas formas de se obter informações que respondam a essa pergunta. Vendedores em lojas físicas, comentários em lojas virtuais e recomendações de amigos, colegas e afins, são as primeiras opções de leigos.

Já para aqueles que têm conhecimento aprofundado das tecnologias que compõem um celular, essas recomendações não são sempre suficientes. Para uma análise mais aprofundada, é preferível recorrer a análises de desempenho ou comparativos entre dispositivos.

Por ser um sistema operacional que vem crescendo com o decorrer do tempo e por ter programas desenvolvidos seguindo o princípio de software livre, ou seja, programa que pode ser copiado, usado, modificado e redistribuído de acordo com as necessidades de cada usuário (FSF, 2004); foi escolhido o Android como sistema operacional que será utilizado neste TCC.

3.2 Escolha da Google

Mantido pela Google, o sistema operacional é aberto, baseado em Linux, dividido em camadas e compatível com diversos dispositivos. A flexibilidade e abertura do sistema fez com que milhares de pessoas contribuíssem para o desenvolvimento da tecnologia Android.

Como dito anteriormente, a Google é responsável por manter o sistema operacional Android, com isso ela desenvolveu um treinamento para desenvolvedores que queiram desenvolver aplicativos para a plataforma.

Por ser a principal empresa responsável pela criação e manutenção no sistema operacional Android, ela foi escolhida como fonte para retirar as técnicas que serão aplicadas no software determinado para obter resultados.

3.3 Escolha do Foco

A Google disponibiliza, em seu site um treinamento para desenvolvedores em Android (GOOGLE, 2013), aulas que descrevem como realizar uma tarefa específica e mostra exemplos de códigos que podem ser utilizados como base para desenvolver um aplicativo.

As aulas estão divididas nos grupos apresentados a seguir:

- **Aula 1** Introdução: ensina o essencial para o desenvolvimento Android, exemplos de atividades, como construir um aplicativo, construção de uma interface de usuários dinâmica e como interagir com outros aplicativos.

- **Aula 2** Aplicativos de construção com compartilhamento de conteúdo: ensina a criar aplicativos que podem compartilhar dados entre dispositivos e aplicativos.

- **Aula 3** Criação de aplicativos com multimídia: ensina a criar aplicações ricas em multimídia que se comportam de acordo com a expectativa do usuário. Exemplos de atividades: gerência de reprodução de áudio e captura de fotos.

- **Aula 4** Construção de aplicativos com gráficos e animação: ensina a construção de uma interface mais robusta, com a realização de algumas tarefas que podem fazer a diferença na concorrência com outros aplicativos. Exemplos de atividades: exibir mapas de bits de forma eficiente e adicionar animações.

- **Aula 5** Construção de aplicativos com conectividade em nuvem: ensina como conectar a outros dispositivos, conectar-se a internet, backup e sincronização de dados, por exemplo, conexão de dispositivos sem fio, executar operações de rede, sincronizar para a nuvem, resolvendo conflitos salvos na nuvem.

- **Aula 6** Construção de aplicativos com informações de usuário e localização: ensina como adicionar a personalização do usuário ao aplicativo. Exemplos de atividades: acessar dados conectados.

- **Aula 7** Melhores práticas para interação e engajamento: ensina a manter os usuários através de implementação de padrões de interação em Android. Exemplos de atividades: planejar e implementar navegação eficaz, notificar usuário, adicionar funcionalidade de pesquisa.

- **Aula 8** Melhores práticas para interface de usuário: ensina construir uma interface usando layouts Android para todos os tipo de aplicativos. Exemplos de atividades: projetar para múltiplas telas, projetar para televisão, implementar acessibilidade e criar interfaces personalizadas.

- **Aula 9** Melhores práticas para a entrada do usuário: ensina a gerenciar *touch screen* e entrada de textos. Exemplos de atividades: usar gestos de toque, manuseio de entrada de teclado, apoiar controladores de jogo.

- **Aula 10** Melhores práticas para serviços de fundo: ensina a executar procedimentos para aumentar o desempenho do aplicativo e diminuir o consumo da bateria. Exemplos de atividades: executando em um serviço de fundo e carregando dados em segundo plano.

- **Aula 11** Melhores práticas para obter aumento no desempenho: ensina a desenvolver um aplicativo mais leve, rápido, menor consumo possível de memória e de bateria. Exemplos de atividades: gerenciar a memória do aplicativo, dicas de desempenho, melhorar o desempenho do layout, otimizar a vida da bateria.

- **Aula 12** Melhores práticas para a segurança e privacidade: ensina como manter seguro os dados do aplicativo. Exemplos de atividades: dicas de segurança, segurança com HTTPS e SSL.

- **Aula 13** Melhores práticas para teste: ensina maneiras eficientes de testar o aplicativos. Exemplo de atividade: teste a aplicação.

Este trabalho focará em analisar as melhores práticas para o desempenho, focando nas técnicas de programação que podem afetar o desempenho de uma aplicação Android, ou seja, concentrará no conteúdo referente a Aula 11, em especial, nas atividade relacionada a dicas de desempenho.

Foi escolhido esse foco, pois o desenvolvimento para dispositivos móveis tende a possuir restrições. Com isso a aplicação das técnicas de programação pode ser o diferencial no desempenho de uma aplicação Android. Além disso, é a única aula que contém dados numéricos que podem ser utilizados como parâmetros de análise deste trabalho.

3.4 Dicas de Desempenho

Nesta etapa serão detalhadas as quatro técnicas de programação que foram selecionadas através do critério de seleção: técnicas que apresentam dados numéricos de ganho de desempenho ao serem utilizadas.

3.4.1 Utilizar estática ao invés de virtual

Esta técnica afirma que, caso seja preciso obter o valor de campos de um objeto, deve-se fazer o uso do método estático. Essa forma de trabalhar é considerada uma boa

prática de programação, pois a assinatura do método que fizer a chamada a este método estático não poderá alterar o estado do objeto.

De acordo com a Google, essa boa prática afeta o desempenho de uma aplicação, fazendo com que ela fique de 15% a 20% mais rápida.

3.4.2 Evite Getters / Setters internos

A utilização de getters e setters é muito comum nas linguagens orientadas a objetos, pois implementa os conceitos de encapsulamento. Devido a isso, é quase que obrigatória a utilização de métodos para atribuir ou recuperar o valor do atributo de um objeto. Porém essa é uma ideia ruim para Android, pois métodos são mais custosos que acessos diretos a atributos de objetos.

De acordo com a Google, a utilização de acessos diretos aos atributos pode ser de três até sete vezes mais rápido que a utilização de métodos para se obter o valor do atributo. Portanto, a utilização dessa técnica pode ter uma alteração considerável no desempenho da aplicação, e por isso foi selecionada para este projeto de pesquisa.

A Figura 4 apresenta um trecho de código com Getter e Setter.

Figura 4 – Com Getter e Setter

```
private int atributo;

public int getAtributo() {
    return atributo;
}

public void setAtributo(int atributo) {
    this.atributo = atributo;
}
```

```
classe.setAtributo(1);
int atributo = classe.getAtributo();
```

O mesmo trecho de código é apresentado na Figura 5 sem Getter e Setter.

Figura 5 – Sem Getter e Setter

```
public int atributo;
```

```
classe.atributo = 1;
int atributo = classe.atributo;
```

3.4.3 Usar sintaxe aprimorada da estrutura de repetição for

A estrutura de repetição *for* é muito utilizada ao se programar e tem a varredura de listas como uma de suas principais utilizações. Na maioria das linguagens de programação, a maneira como a lista é percorrida não influencia o desempenho de maneira significante, porém para Android esse fato não é verdadeiro, e a utilização de um *for* aprimorado, também conhecido como *for each*, pode alterar o desempenho de uma aplicação.

O *for* aprimorado pode ser utilizado para percorrer coleções que implementam a interface Iterable. Sua utilização, de acordo com a Google, pode ser três vezes mais rápida que a utilização de um índice para controlar os acessos à lista.

A Figura 6 apresenta um exemplo de código sem uso do *for* aprimorado.

Figura 6 – Sem *for* aprimorado

```
static class classe() {

    int atributo;
}

Classe[] lista = new Classe[1];

public void funcao(){
    int soma = 0;
    for (int i = 0; i< lista.length; i++){
        soma += lista[i].atributo;
    }

}
```

A Figura 7 apresenta um exemplo de código com uso do *for* aprimorado.

Figura 7 – Com *for* aprimorado

```
static class classe() {

    int atributo;
}

Classe[] lista = new Classe[1];

public void funcao(){
    int soma = 0;
    for (Classe classe : lista){
        soma += classe.atributo;
    }

}
```

A grande diferença entre os codigos das Figuras 6 e 7, é que o uso do *for* aprimorado faz com que a utlização do índice como controle de acesso à lista seja descartado.

3.4.4 Evitar o uso de ponto flutuante

De acordo com a Google, ao usar tipo de dado de ponto flutuante em varáveis, uma aplicação Android fica cerca de duas vezes mais lento do que o uso número inteiro. Dessa forma, recomenda-se não usá-los sempre que possível.

A Figura 8 apresenta um exemplo de código sem uso de ponto flutuante.

Figura 8 – Sem uso de ponto flutuante

```
public int valor = 1;
```

A Figura 9 apresenta um exemplo de código com uso de ponto flutuante:

Figura 9 – Com uso de ponto flutuante

```
public float valor = 2.00f;
```

4 Ferramentas e Software

Este capítulo mostrará o protocolo de pesquisa definido para buscar ferramentas para a coleta de dados no contexto dessa pesquisa. Elas serão utilizadas para coletar dados, além de definir o aplicativo escolhido para ser utilizado na aplicação das técnicas de programação definidas no Capítulo 3.

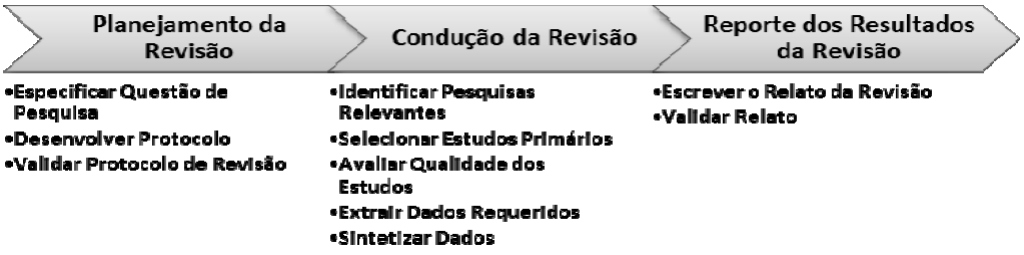
4.1 Protocolo de Pesquisa

Foi utilizada a revisão sistemática como inspiração para definir o método de pesquisa para encontrar ferramentas de coleta de dados sobre desempenho em aplicativos desenvolvidos utilizando o sistema operacional Android.

A técnica de revisão sistemática consiste em um explícito e rigoroso método para identificar, criticar e sintetizar estudos relevantes de um tópico em particular (BRERETON et al., 2006).

Os procedimentos gerais para realizar uma revisão sistemática em engenharia de software são: planejamento da revisão, em que ocorre a identificação das necessidades para a revisão e elaboração do protocolo; condução da revisão, em que ocorrem a identificação da pesquisa, a seleção e avaliação da qualidade dos estudos, e a extração e síntese dos dados; e reporte dos resultados da revisão, que permite que leitores possam criticar e replicar a revisão (BRERETON et al., 2006). A Figura 10 ilustra este processo.

Figura 10 – Processo de Revisão Sistemática (BRERETON et al., 2006)



Como dito, o processo de Revisão Sistemática, definido na Figura 17, não foi executado em sua totalidade. A Tabela 3 mostra o que foi Realizado, o que foi Realizado Parcialmente e o que Não foi Realizado.

Tabela 3 – Resumo adaptado do Processo de Revisão Sistemática

Realizado	**Realizado Parcialmente**	**Não foi Realizado**
Especificar Questão de Pesquisa	Escrever o Relato da Revisão	Validar Protocolo de Revisão
Desenvolver Protocolo	Extrair Dados Requeridos	Avaliar Qualidade dos Estudos
Identificar Pesquisas Relevantes		Sintetizar Dados
Selecionar Estudos Primários		Validar Relato

4.1.1 Objetivo e Questões de Pesquisa

O objetivo desta pesquisa é analisar relatos de ferramentas utilizadas para coletar dados referentes ao desempenho em aplicações Android com propósito de selecionar ferramentas que podem ajudar a coletar dados referentes ao desempenho no contexto de desenvolvimento Android.

A partir do objetivo estabelecido anteriormente, foi elaborada a seguinte questão de pesquisa:

- Q1. Quais são as ferramentas mais utilizadas para coleta de dados referentes a desempenho em aplicações desenvolvidas utilizando o sistema operacional Android?

4.1.2 Processo e Critérios de Seleção

A estratégia adotada é composta por três etapas:

- Identificação das fontes: Consiste na seleção das fontes para se realizar as pesquisas manuais e automáticas.
- Definição da string de pesquisa: Realizada de maneira objetiva.
- Condução da pesquisa automatizada: Utiliza as strings para realizar as pesquisas na fonte.

Para aplicações das strings de busca foi selecionada dentre as bibliotecas digitais o *IEEEXplore* para a busca dos artigos, por ser uma máquina de busca específica para Engenharia de Software.

Os critérios para definir se resultados encontrados serão selecionados são:

1. Os artigos devem utilizar a língua inglesa.
2. Os artigos disponíveis para download ou leitura de forma gratuita.
3. Os artigos devem apresentar estudo sobre ferramentas ou qualquer outro item relacionados ao desempenho em aplicações Android.

Os critérios para definir se resultados encontrados serão excluídos são:

1. O artigo trate o tema de maneira superficial.
2. O artigo não utilize ferramentas para a coleta de dados sobre desempenho em aplicações Android.

4.1.3 Procedimento para Extração dos Dados

Como estratégia de extração de informações, considerou-se que para cada artigo selecionado, serão extraídas as seguintes informações: título, autor(es), data de publicação e a informação da ferramenta.

4.1.4 Condução da Pesquisa

Para realizar a pesquisa foi definida a string de busca criada a partir do objetivo desta pesquisa bibliográfica.

Figura 11 – String de busca definida

```
("android" AND "performance")
```

Muitos artigos foram retornados pela execução da string de busca da Figura 18, por isso foi feito um refinamentos na string de busca para encontrar os artigos mais próximos do tema abordado nesse TCC. O resultado final pode ser observado na Figura 19.

Figura 12 – String de busca final definida

```
(Android AND performance AND (tool OR application OR app OR software) AND measurement)
```

O resultado da execução da string de busca:

Tabela 4 – Resultado da Pesquisa

STRING DE BUSCA	FONDE DE BUSCA	RESULTADOS
Inicial	IEEEXplore	419
Final		32

Dos 32 artigos analisados, apenas 15 artigos mostram alguma ferramenta para coletar dados sobre desempenho em aplicações Android.As ferramentas retornadas por essa pesquisa foram: DDMS e Little Eye.

4.2 Ferramentas

Esta seção mostra o detalhamento das ferramentas retornadas pela pesquisa, juntamente com suas respectivas dependências.

4.2.1 DDMS

Os dispositivos Android vêm nativamente com uma ferramenta de depuração chamada Dalvik Debug Monitor Server (DDMS), ou Servidor de Monitoramento de Debug do Dalvik, que fornece serviço de captura de tela, informações de Heap e Threads do dispositivo, estado do LogCat, processos e intensidade do sinal recebido pela antena do dispositivo, e ainda coleta informações de chamadas, mensagens de texto e dados de localização (DEVELOPER, 2013).

No Android, cada aplicativo é executado em seu próprio processo, cada um dos quais é executado em sua própria Máquina Virtual (VM). Cada VM expõe uma única porta que um depurador pode anexar. Quando DDMS é iniciado, ele se conecta ao Android Debug Bridge (ADB). Quando um dispositivo for conectado ao computador, um serviço de monitoramento de VM é criada entre ADB e o DDMS, que notifica o DDMS quando uma VM no dispositivo é iniciado ou encerrado. Uma vez que a VM está rodando, o DDMS recupera ID do processo da VM (PID), via ABD, e abre uma conexão com depurador do VM, através do daemon do ABD (ADBD) do dispositivo. DDMS pode agora falar com a VM usando um protocolo personalizado (DEVELOPER, 2013).

Abaixo são mostradas as ferramentas que deverão ser instaladas para sua utilização:

- Software Development Kit (SDK) do Android: Fornece aos desenvolvedores todas as ferramentas para desenvolver um aplicativo para Android, desde o ambiente de desenvolvimento, até ferramentas para gerenciar o download e instalação de novos

pacotes e configurações para as versões mais atuais do Android (DEVELOPER, 2013).

- Android Developer Tools (ADT), ou Ferramentas para o Desenvolvedor Android. É um plugin para o Eclipse (ECLIPSE, 2013) que integra nesta IDE todas as ferramentas do SDK. Ele oferece vários recursos que ajudam a desenvolver aplicações para Android rapidamente e com segurança. Além da integração com o Eclipse, que é uma IDE bem estabelecida e conhecida pelos desenvolvedores, o ADT oferece ferramentas visuais para diversos comandos do SDK, que geralmente seriam acessados somente através de linhas de comando

- Android Debug Bridge (ADB) é uma ferramenta de linha de comando versátil que permite a comunicação com uma instância do emulador ou dispositivo Android conectado ao computador via USB.

- O sistema de Log Android fornece um mecanismo para coleta e visualização de saída de depuração do sistema, registros de várias aplicações e partes do sistema são recolhidos numa série de buffers circulares, que podem então ser visualizados pelo comando *logcat* (DEVELOPER, 2013).

4.2.2 Little Eye

Little Eye é uma ferramenta de análise de desempenho e monitoramento que pode ajudar a identificar e corrigir problemas de desempenho em uma aplicação Android com versões a partir da 2.3 (RANGARAJAN et al., 2013).

Esta ferramenta apoia a coleta de métricas sobre CPU, recursos de rede, memória RAM, consumo de energia, GPS e espaço em disco. Tem como principais características (RANGARAJAN et al., 2013):

- **Medir:** mede o desempenho e consumo de recursos do aplicativo. Obtém estatísticas detalhadas sobre a quantidade de energia, dados, CPU, memória e espaço em disco que o aplicativo está utilizando.

- **Analisar:** informa detalhadamente o contexto em torno do aplicativo e os dados coletados, possibilitando uma compreensão muito mais profunda do comportamento do aplicativo.

- **Otimizar:** obtém rapidamente acionáveis "próximos passos" para otimizar o consumo de recursos do aplicativo, seja problemas de memória, bateria, uso excessivo de dados ou até mesmo falhas frequentes.

Para a instalação desta ferramenta, é necessário (RANGARAJAN et al., 2013):

1. Java JRE ou SDK - V 1.6 + (Java 6 ou superior).
2. SDK Android.
3. Depuração USB está ativada no telefone.
4. Se você estiver usando o Windows, pode ser necessário para garantir que você está se conectando como um "Camera (PTP)" e não como um "dispositivo de mídia (MTP)".
5. Se você estiver usando o Windows, você pode precisar instalar os drivers do seu telefone.

4.3 Software

O software escolhido para ser alterado de acordo com as técnicas de programação testadas deveria ser um software livre, ou seja, programa que pode ser copiado, usado, modificado e redistribuído de acordo com as necessidades de cada usuário (FSF, 2004).

Para procurar esses possíveis softwares foi utilizado o *Git Hub* (HUB, 2014), um site no qual é possivel hospedar softwares, podendo ser compartilhado com outros usúarios.

4.3.1 CPU Spy

O CPU Spy é um aplicativo simples utilizado para exibir o tempo que a CPU gasta em cada estado de frequência. Esta pode ser uma ferramenta útil no diagnóstico de problemas de bateria ou aprimorando suas configurações, também exibe as informações do kernel atual.

Para ter acesso ao código fonte deste aplicativo, basta baixar o artigo disponibilizado no endereço:<https://github.com/bvalosek/cpuspy>.

Este aplicativo foi escolhido, pois possibilita a aplicação de todas as técnicas de programação que foram selecionadas no Capítulo 3.

5 Métricas de software

De forma geral, pode-se dizer que medição é o processo pelo qual símbolos ou números são associados a atributos do mundo real de forma a descreve-los de acordo com as regras definidas (BERGHOUT; SOLINGEN, 1999). Segundo esses mesmos autores, medição em software é um processo contínuo de definir, coletar e analisar dados com respeito ao processo de software e os seus produtos. A principal razão para o uso de medições é o fato de esperar que produzam um entendimento sobre o processo de software que o torne controlável, além disso, fornecer informações que permitem aperfeiçoá-lo.

Neste capítulo serão apresentadas as abordagens Goal Question Metrics (GQM) e Practical Software Measurement (PSM) que são as mais utilizadas para definição de métricas na Engenharia de Software. Em seguida, é apresentada a abordagem escolhida juntamente com os argumentos que levaram à escolha.

5.1 Goal Question Metric

GQM é uma abordagem definida por Victor Basili (BASILI; CALDIERA; ROMBACH, 1994) utilizada como guia na definição de métricas para um processo de mensuração. Ela foi originalmente desenvolvida para avaliações de defeitos de um conjunto de projetos no ambiente da NASA, *Goddard Space Flight*, no início da década de 80.

Para o uso do GQM, deve-se proceder da forma *top-down*, no qual inicialmente são identificados os objetivos que precisam ser atingidos. Diante desses objetivos definidos, um conjunto de questões é definido e por fim são criadas as métricas que satisfaçam os objetivos. Vários modelos GQM também podem possuir questões e métricas em comum, mas é importante lembrar que os pontos de vista podem mudar, portanto, devem ser levados em consideração nas medições (BASILI; CALDIERA; ROMBACH, 1994).

A Figura 13 ilustra a hierarquia de objetivos, questões e métricas apresentadas anteriormente.

Figura 13 – GQM Hierarquia (BASILI; CALDIERA; ROMBACH, 1994)

5.2 Practical Software Measurement

PSM é um modelo para mensuração de projetos de software criado em 1994, sob o patrocínio do Departamento de Defesa Norte Americano. Em 1997 foi determinado um projeto para construção de uma padrão ISO (Organização Internacional para Padronização), o qual resultou na norma ISO/IEC 15939, na qual o PSM foi utilizado como base (ISO/IEC, 2007).

O processo de medição proposto pela ISO/IEC 15939 é orientado às necessidades de informação da organização. Para cada necessidade de informação, o processo gera um produto de informação para sanar a necessidade.

O PSM é um modelo que estrutura medições em atividades de projetos de software. Nesse sentido, objetiva medir e melhorar processos, projetos e produtos de software. Em nível prático o PSM aborda dois problemas:

- Como especificar de uma maneira formal as métricas que deverão ser utilizadas no projeto, ou seja, especificar formalmente os indicadores a serem usados.
- Como deverá ser conduzido o processo de medição.

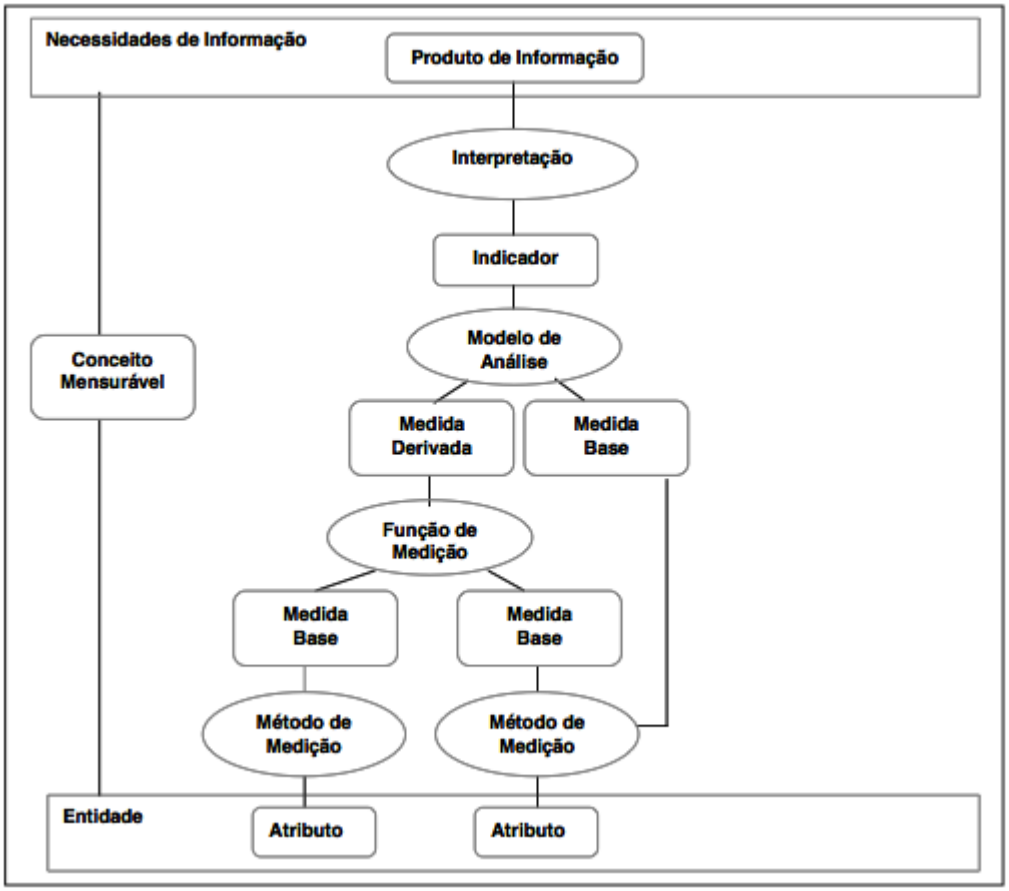
Para atingir tais objetivos, o PSM faz uso do modelo de informação e o modelo de processo, os quais são detalhados nas subseções seguintes.

5.2.1 Modelo de Informação

O Modelo de Informação do PSM é uma estrutura para a definição das medidas que deverão ser utilizadas no projeto (ver Figura 14). Para isso, o PSM trabalha com o conceito de necessidades de informações mensuráveis que, por sua vez, resultam de esforços dos

gerentes para influenciar as saídas de projetos, processos e iniciativas oriundas de objetivos de medição. As necessidades de informações são usualmente derivadas de duas fontes: objetivos que os gerentes procuram atingir em seus projetos e obstáculos que impedem de atingir tais objetivos (MCGARRY et al., 2001).

Figura 14 – Modelo de Informação de Medição (ISO/IEC, 2007)



5.2.2 Modelo de Processo

O modelo de processo serve como guia a implementação do PSM. É o processo para o estabelecimento, planejamento, execução e avaliação da medição do projeto, organização ou empreendimento como um todo (ISO/IEC, 2007).

A seguir são apresentadas as cinco perspectivas que são centrais para o processo de medição:

1. **Desempenho**: refere-se aos valores característicos que são observados quando medidos os atributos dos produtos e serviços, advindos do processo. Pode ser descrito de duas formas: ao se medir os atributos que os processos produzem e ao se medir os atributos do processo em si.

2. **Estabilidade**: refere-se à habilidade de uma organização em produzir produtos de acordo com um plano e em melhorar processos visando produtos melhores e mais competitivos.

3. **Conformidade**: refere-se ao processo estar claramente definido, efetivamente suportado, fielmente executado e reforçado.

4. **Capacidade**: refere-se aos valores e características que irão variar conforme o passar do tempo. Quando um processo é estável, este possuirá meios previsíveis dentro de intervalos previsíveis. Então é possível examinar se o processo está ou não atingindo os objetivos.

5. **Melhora e Investimento**: refere-se a objetivos de negócio e estratégias, os quais juntamente com dados factuais sobre os atributos de qualidade do produto e desempenho do processo, são as chaves que levam a ações que aperfeiçoam o processo de software.

A Tabela 5 mostra as sete categorias de informação definidas pelo PSM com seus respectivos conceitos mensuráveis que são uma ideia sobre as entidades que poderiam ser medidas para satisfazer uma necessidade identificada.

Tabela 5 – Categoria da informação e conceito mensurável (MCGARRY et al., 2001)

CATEGORIA DA INFORMAÇÃO	CONCEITO MENSURÁVEL
Cronograma e Progresso	Alcance dos Marcos do Projeto
	Desempenho do Caminho Crítico
	Progresso da Unidade de Trabalho
	Capacidade Incremental
Recurso e Custo	Esforço da Equipe
	Desempenho Financeiro
	Recursos Ambientais e de Suporte
Tamanho e Estabilidade do Produto	Tamanho e Estabilidade Físicos
	Tamanho e Estabilidade Funcionais
Qualidade do Processo	Corretude Funcional
	Facilidade de Manutenção
	Eficiência
	Portablidade
	Usabilidade
	Confiabilidade
Desempenho do Processo	Flexibilidade do Processo
	Eficiência do Processo
	Efetividade do Processo
Efetividade da Tecnologia	Adequabilidade da Tecnologia
	Volatilidade da Tecnologia
Satisfação do Cliente	Feedback do Cliente
	Suporte ao Cliente

O PSM aborda o chamado construtor de medição, que é uma estrutura que relaciona atributos específicos de processos e/ou produtos, medidos com uma informação específica necessária. O construtor de medição define três níveis de medida: medida básica, medida derivada e indicador. Cada um desses níveis de medida acrescenta regras específicas para determinar valores e definir métodos, funções e modelos para a medição (MCGARRY et al., 2001).

5.3 Considerações Finais

Para este trabalho será adotada o *Goal Question Metric* (GQM), pois as medições serão orientadas a objetivos e por ter uma metodologia simples e eficaz que atende melhor os objetivos deste trabalho. Além disso, o GQM é um método altamente reconhecido pela academia, tendo sido explorado e experimentado intensamente ao longo dos anos de estudos em engenharia de software e que ele satisfaz as necessidades de medições identificadas neste trabalho.

6 Experimento

Após a escolha das técnicas de programação no Capítulo 3; das ferramentas para coleta e do software, no Capítulo 4; da abordagem GQM, no Capítulo 5; foi realizado um conjunto de medições cujo delineamento baseou-se na técnica de experimento que passará por quatro etapas descritas a seguir:

- Definição dos objetivos
- Planejamento
- Operação
- Análise e Interpretação

6.1 Definição dos Objetivos

Esta seção explora os objetivos de experimento em termos de: objetivo geral e das métricas estabelecidas.

6.1.1 Objetivo Global e Foco de Medição

Motivado pela necessidade de aplicativos cada vez mais eficientes, pelo crescente número de dispositivos móveis utilizando o sistema operacional Android e pela quantidade limitada de recursos desses dispositivos, o objetivo deste experimento é avaliar o impacto de um conjunto de técnicas de programação no desempenho de um aplicativo desenvolvido para Android, a fim de encontrar indícios de qual a melhor técnica de programação a ser utilizada visando um melhor desempenho dos recursos CPU e memória .

A Tabela 6 apresenta o objetivo deste experimento no formato sugerido por (WOHLIN et al., 2012).

Tabela 6 – Objetivo de Estudo

ANALISAR:	Técnica de programação
COM O PROPÓSITO DE:	Avaliar
COM RELAÇÃO A:	CPU e memória
DO PONTO DE VISTA:	do usuário
NO CONTEXTO DE:	Fábrica de Software

O foco da medição realizada neste trabalho é o impacto do uso de técnicas de programação nos recursos dos dispositivos móveis citados anteriormente, a fim de verificar qual a melhor técnica de programação entre as definidas na Seção 3.4 do Capítulo 3.

6.1.2 Definição da Hipótese

Foram definidas duas hipóteses para esse experimento:

H0: Pelo menos uma das técnicas de programação *não afeta* a utilização do recurso CPU e/ou memória dos dispositivos móveis.

H1: Pelo menos uma das técnicas de programação *afeta positivamente* a utilização do recurso CPU e/ou memória dos dispositivos móveis.

H2: Pelo menos uma das técnicas de programação *afeta negativamente* a utilização do recurso CPU e/ou memória dos dispositivos móveis.

Neste experimento existe um fator, a técnica de programação, e quatro tratamentos representados pelas técnicas de programação: utilizar estática ao invés de virtual; evitar getters/setters internos; usar sintaxe aprimorada da estrutura de repetição for; evitar o uso de ponto flutuante.

As hípoteses definidas anteriormente foram usadas para elaboração das questões que serão apresentadas na seção seguinte.

6.1.3 Questões e Métricas

A transformação de objetivos em questões estabelece e aperfeiçoa a medição de metas. Ao responder as perguntas, deve-se ser capaz de concluir se o objetivo foi alcançado. A pergunta foi derivada do objetivo apresentado na Tabela 6 e serviu de base para a definição das métricas para cada objetivo definido.

Q1: Como calcular o impacto do uso de aplicativos nos recursos CPU e memória dos dispositivos móveis?

M1: Consumo de Memória

M2: Consumo de CPU

6.1.4 Detalhamento das Métricas

As métricas M1-Consumo de Memória e M2-Consumo de CPU foram detalhadas para facilitar a realização da coleta e análise dos dados. As Tabelas 7 e 8 apresentam cada uma delas

Tabela 7 – Consumo de memória

MÉTRICA	M1 – Consumo de memória
DESCRIÇÃO	Quantidade máxima de memória utilizada pela aplicação
FÓRMULA	Quantidade utilizada pela aplicação = Total - Quantidade de memória livre
TIPO DE ESCALA	Racional
ESTIDADES E ATRIBUTOS	**Entidade:** O software , **Atributos:** Quantidade de memória
TIPO DE MEDIÇÃO	Direta
CLASSIFICAÇÃO DA MEDIDA	Objetiva
FAIXA DE VALORES	0 a 100
TIPO DE COLETA	Automática
COLETA	**Procedimento:** coleta automatizada, **Documentação:** gráfico de uso da memória fornecido pelo software de coleta de métricas
ANÁLISE	**Responsável pela Análise:** Pesquisador, **Periodo de Análise:** antes e após a implementação de uma técnica
APRESENTAÇÃO	**Forma de apresentação:** gráfica, **Interessado pela Informação:** Pesquisador
UNIDADE DE MEDIDA	megabyte

Tabela 8 – Consumo de CPU

MÉTRICA	M2 – Consumo de CPU
DESCRIÇÃO	Percentual de uso de CPU pela aplicação
FÓRMULA	Quantidade utilizada pela aplicação = Total - Quantidade de CPU livre
TIPO DE ESCALA	Racional
ESTIDADES E ATRIBUTOS	**Entidade:** O software , **Atributos:** % de uso da CPU
TIPO DE MEDIÇÃO	Direta
CLASSIFICAÇÃO DA MEDIDA	Objetiva
FAIXA DE VALORES	0 a 100
TIPO DE COLETA	Automática
COLETA	**Procedimento:** coleta automatizada, **Documentação:** gráfico de uso do CPU fornecido pelo software de coleta de métricas
ANÁLISE	**Responsável pela Análise:** Pesquisador, **Periodo de Análise:** antes e após a implementação de uma técnica
APRESENTAÇÃO	**Forma de apresentação:** gráfica, **Interessado pela Informação:** Pesquisador
UNIDADE DE MEDIDA	%

As Tabelas 7 e 8 foi baseada no template para detalhamento das métricas apresentado por (RAMOS, 2013).

6.2 Planejamento

Nesta seção será detalhado o planejamento utilizado para a realização desde experimento definido em: seleção do contexto, seleção dos indivíduos, seleção das variáveis, instrumentação e validade.

6.2.1 Seleção do Contexto

De acordo com (WOHLIN et al., 2012), o contexto deste experimento pode ser caracterizado por duas dimensões:

- REALIDADE: o problema real
- GENERALIDADE: específico

A realidade é o problema real, pois as técnicas de programação são caracterizadas durante o problema de desempenho. As técnicas de programação serão testadas na aplicação móvel CPU Spy a qual utiliza o sistema operacional Android, por isso, o contexto possui o caráter específico.

6.2.2 Seleção das Variáveis

Variável independente: Técnicas de programação.

Variável dependente: Consumo de CPU e consumo de memória dos dispositivos móveis relacionados ao desempenho das aplicações que utilizam o sistema operacional Android.

6.2.3 Descrição da Instrumentação

Para o teste das hipóteses deste experimento, foi utilizado o software CPU Spy, que é um sistema de monitoramento do uso do CPU em cada estado de frequência, e a ferramenta Little Eye, que é a única ferramente dentre as retornadas na Seção 4.2 no Capítulo 4 que fornece dados referentes a consumo de CPU e memória.

Para a execução do experimento, fez-se necessário modificar o código do CPU Spy incluindo as técnicas recomendadas pela Google. Todas as mudanças que foram feitas no código do CPU Spy para garantir a execução do experimento podem ser vistas nos Apêndices **B** e **C** deste trabalho.

Foram coletados os valores referentes ao desempenho levando em consideração o Consumo de CPU e Consumo de Memória antes e após a implementação das técnicas de programação selecionadas no Capítulo 3.

As configurações de hardware e software do smartphone e do computador utilizados para realização do experimento foram as seguintes:

- Smartphone: Samsung Galaxy SIII(I9300)
  - Configurações de Hardware:
    - Processador: Quad-core 1.4 GHz Cortex-A9
    - Memória: 16 GB
    - Memória RAM: 1 GB
  - Configurações de Software
    - Android: 4.3 (Jelly Bean)
- Notebook Samsung NP540
  - Configurações de Hardware:
    - Processador: Intel Core i3-370M
    - Memória: 4 GB DDR2
    - HD: 500 GB
    - Placa de Vídeo: ATI Mobility Radeon HD 545v

- Configurações de Software
  - Sistema Operacional: Microsoft Windos xP Professional
  - IDE: Eclipse Java Development Tools 3.8.2
  - Android: 4.4 (API 19)
  - Pacote Java : Java Development Kit 1.7

Tanto o computador quanto o smartphone foram formatados. Para realizar as medições desse experimento foram instalados respectivamente em casa um deles o Little Eye e o CPU Spy, além disso ambos estavam conectados a uma fonte de alimentação.

O software escolhido para ser o piloto deste trabalho foi o CPU Spy, a justificativa para a escolha do aplicativo esta na Seção 4.3.1 do Capítulo 4.

6.2.4 Validade

As fraquezas deste experimento serão analisadas sob os seguintes aspectos:

Validade interna: diz respeito a assuntos que podem afetar a variável independente no que diz respeito à causalidade, sem o conhecimento do pesquisador. O experimento foi executado por uma única pessoa, pois os aplicativos são executados em aparelhos móveis indepententes, sendo assim, a quantidade de pessoas que executa não interfere no resultado deste experimento. Pórem existe uma ameaça relacionada a instrumentação, pois as configurações de hardware e de software dos dispositivos utilizados podem influenciar nos resultados do experimento. Para mitigar essa ameaça tanto o computador quanto o smartphone foram formatados. Outra ameaça encontra é o fato do processo de operação do experimento ser repetitido causando fadiga podendo afetar os dados obtidos.

Validade Externa: diz respeito a generalização da experiência resultar em outros ambientes do que aquele em que o estudo é realizado. Como foi mencionado na parte "Validade interna" o participante do estudo podem ser considerado representativo para a população de usuários de aplicativos móveis. Porém o uso de outro aplicativos rodando simultaneamente que podem interferir nos valores resultantes o que representa uma fraqueza no resultado. Para mitigar essa ameaça foram retirados todos os aplicativos do smartphone deixando apenas o CPU Spy que é o aplicativo analisado.

Validade de Construção: diz respeito a preocupação entre teoria (causa e efeito) e observação (tratamentos e saídas). O experimento mostra o efeito que as técnicas de programação causam nos recursos dos dispositivos móveis CPU e memória. Uma ameaça para essa validade é o fato de as técnicas de programação selecionadas não serem suficientes para garantir que uma determinada técnica de programação e a melhor para o recurso do dispositivo móvel em questão. Para mitigar essa ameaça foram escolhidas as técnicas de programação citadas pela empresa mantenedora do sistema operacional Android.

Validade de Conclusão: diz respeito à análise estatística dos resultados. Neste experimento foram utilizadas a moda e a média para fazer uma conclusão sobre as técnicas de programação. Porém uma ameaça para essa validade é o fato de ter poucas amostras o que pode acabar reduzindo a capacidade de revelar padrões nos dados. Para mitigar essa ameaça foi analisada a variação de cada técnica de programação durante a medição para garantir que não teve nenhuma variação que contribuísse para que os dados não fossem desconsiderados.

6.3 Operação

A coleta de dados foi realizada de forma automatizada, por meio da ferramenta descrita no Capítulo 4. No entanto, identificou-se também a necessidade de um controle manual do tempo de monitoramento executado pela ferramenta Little Eye, pois tal monitoramento exige que o usuário clique nos botões para iniciar e parar a atividade. Para as medições foram executados os seguintes passos:

1. Aos trinta segundos, executar aplicação com a ferramenta Little Eye
2. Aos dois minutos, executar as funcionalidades Reset Timers e Refresh em sequência
3. Aos dois minutos e trinta segundos, executar a funcionalidade Refresh
4. Aos três minutos, executar a funcionalidade Refresh
5. Aos três minutos e trinta segundos, executar as funcionalidades Reset Timers e Refresh em sequência
6. Aos quatro minutos, executar a funcionalidade Refresh
7. Aos quatro minutos e trinta segundos executar a funcionalidade Refresh
8. Aos cinco minutos, parar monitoramento com a ferramenta Little Eye

O conjunto de passos descritos foi executado dez vezes para cada técnica de porgramação antes da sua implementação e dez vezes após a sua devida implementação. Para cada teste realizado obtém-se o resultado sobre o consumo de CPU e memória. Sendo assim, foram realizados oitenta testes para coletar os dados desse trabalho.

6.4 Análise e Interpretação

Nesta seção será detalhada a análise e interpretação dos resultados realizada neste experimento que é composta por: estatística descritiva, resultados, análise quantitativa e conclusão sobre as hipóteses.

6.4.1 Estatística Descritiva

Como foi utilizado um único valor para representar todos os valores coletados, será feita o uso de medidas de tendência central. Dentro das medidas de tendência central existem vários tipos de medida. Os valores “Consumo de CPU” e “ Consumo de Memória” são escalas ordinais e, por isso, é possível utilizar a “moda“ e “média“.

As Tabelas 9 e 10, mostram a moda e a média antes e após a implementação das técnicas de programação. Para entendê-las utilize a legenda a seguir:

- T1: Utilizar estático ao invés de virtual
- T2: Evite Getters/Setters internos
- T3: Usar sintaxe aprimorada da estrutura de repetição for
- T4: Evitar uso de ponto flutuante

A Tabela 9 mostra o consumo de CPU para cada técnica de Programação antes e após a implementação.

Tabela 9 – Consumo de CPU antes e após a implementação

TÉCNICAS	**ANTES DA IMPLEMENTAÇÃO**		**APÓS A IMPLEMENTAÇÃO**	
	MODA	MÉDIA	MODA	MÉDIA
T1	23	22.7	23	22.7
T2	23	23	23	22
T3	23	22.7	23	24
T4	22	22.4	22	22

Como pode ser visto na Tabela 9, a maioria dos valores tiveram o mesmo resultado, o que permitiu o conhecimento da moda, contribuindo assim para que os dados não fossem desconsiderados. Também foi calculada a média e, ao comparar as médias antes e após a implementação das técnicas de programação, fica claro que houve uma melhora no desempenho do aplicativo analisado com relação ao consumo de CPU nas técnicas de programação T2 e T4.

A Tabela 10 mostra o consumo de memória para cada técnica de Programação antes e após a implementação

Tabela 10 – Consumo de memória antes e após a implementação

TÉCNICAS	**ANTES DA IMPLEMENTAÇÃO**		**APÓS A IMPLEMENTAÇÃO**	
	MODA	MÉDIA	MODA	MÉDIA
T1	34.55	34.551	34.25	34.253
T2	34.57	34.565	34.63	34.618
T3	34.56	34.559	34.15	34.149
T4	34.56	34.556	34.33	34.329

Como pode ser visto na Tabela 10, a maioria dos valores tiveram o mesmo resultado, o que permitiu o conhecimento da moda, contribuindo assim para que os dados não fossem desconsiderados. Também foi calculada a média e, ao comparar as médias antes e após a implementação das técnicas de programação, fica claro que houve uma melhora no desempenho do aplicativo analisado com relação ao consumo de memória nas técnicas de programação T1, T3 e T4.

6.4.2 Resultados

Para coletar dados suficientes para realização do experimento, foram coletadas medidas dez vezes para cada técnica de programação e para cada métrica estabelecida antes e após a implementação da técnica.

Como os valores para cada métrica só variaram a última casa decimal, as Figuras 15 a 22 mostram através dos gráficos essa variação no decorrer das médições onde o eixo X representa número da medição e o eixo Y corresponde a variação da última casa decimal. O detalhamento dos valores das medições estão no Apêndice **A**.

A Figura 15 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas antes da implentação da técnica de programação T1: Utilizar Estático ao Invés de Virtual.

A Figura 16 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas após da implentação da técnica de programação T1: Utilizar Estático ao Invés de Virtual.

Figura 15 – Utilizar estático ao invés de virtual antes da implementação

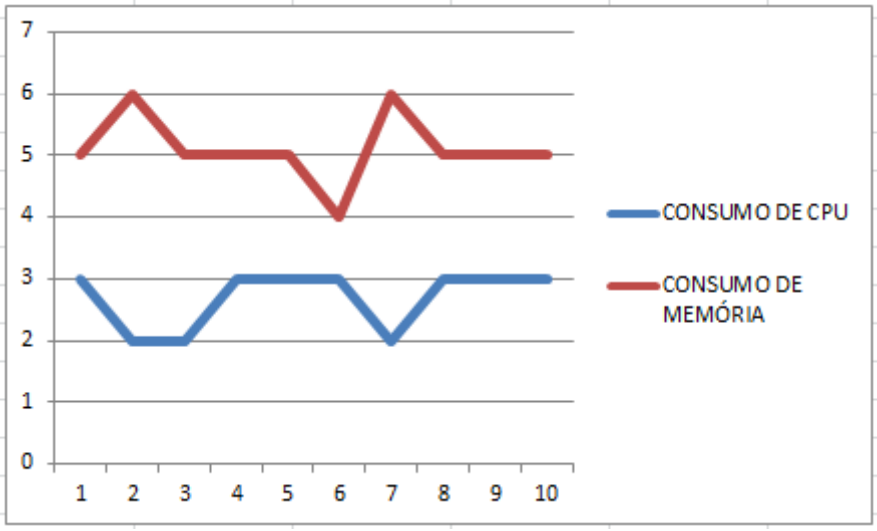
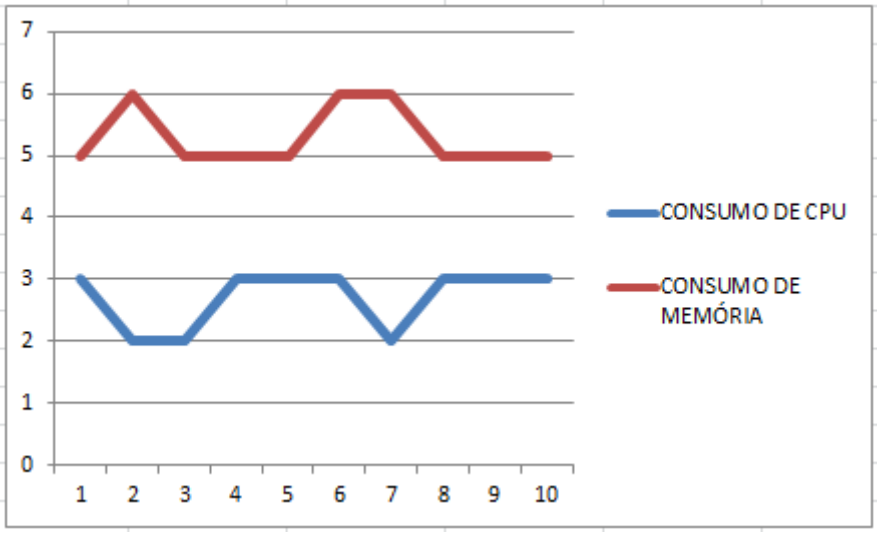


Figura 16 – Utilizar estático ao invés de virtual após da implementação



Analisando a métrica correspondente ao consumo de memória na Figura 15 observou-se uma oscilação de 0,01 para baixo ou para cima em relação à moda da medição. Na Figura 16 verificou-se que essa oscilação ocorreu apenas 0,01 para cima do valor da moda.

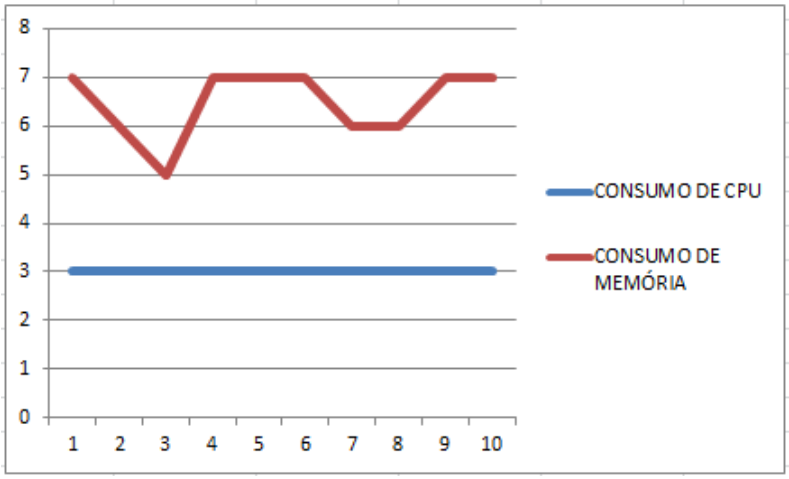
Analisando a métrica correspondente ao consumo de CPU nas Figuras 15 e 16 observou-se uma oscilação de 0,01 para baixo relação à moda da medição.

Com base nas análises de cada métrica nas Figuras 15 e 16 verificou-se que o valor da última casa decimal é repetida durante vários testes o que confirma que os valores não sofreram variações que pudessem desconsiderar os dados coletados. Outra conclusão que pode ser observada a partir dos resultados é que a média dos valores obtidos é bem próxima aos valores coletados, pois estes não sofreram grandes variações.

A Figura 17 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas antes da implentação da técnica de programação T2:

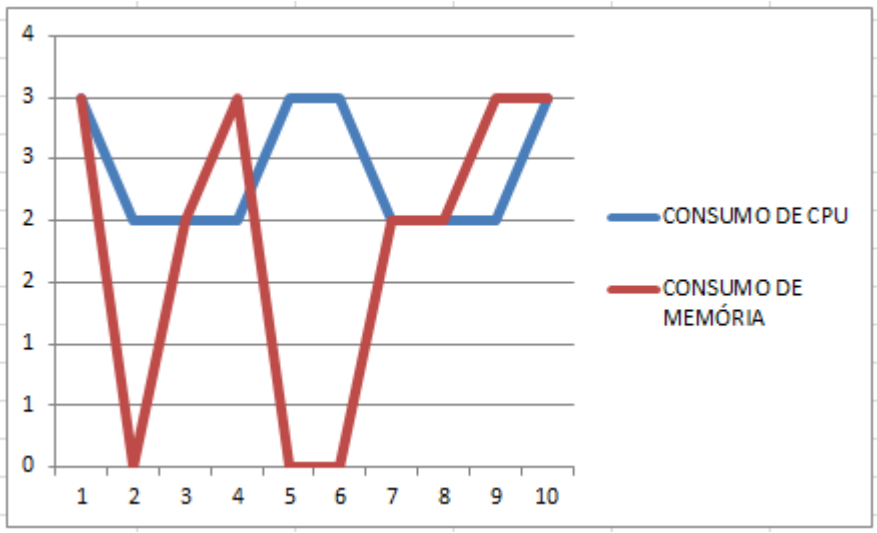
Evite Getters/Setters Internos.

Figura 17 – Evite Getters/Setters internos antes da implementação



A Figura 18 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas após da implentação da técnica de programação T2: Evite Getters/Setters Internos.

Figura 18 – Evite Getters/Setters internos após a implementação



Analisando a métrica correspondente ao consumo de memória na Figura 17 observou-se uma oscilação de até 0,02 para baixo. Na Figura 18 verificou-se que essa oscilação ocorreu de até 0,03 para baixo do valor da moda.

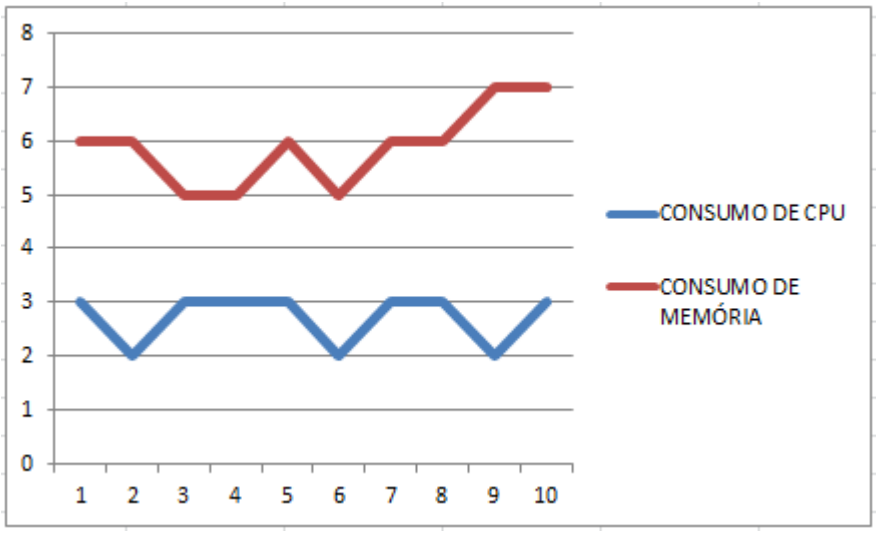
Analisando a métrica correspondente ao consumo de CPU nas Figuras 17 e 18 observou-se oscilação de 0,01 em relação à moda da medição apenas na Figura 18.

Com base nas análises de cada métrica nas Figuras 17 e 18 verificou-se que o valor da última casa decimal é repetida durante vários testes o que confirma que os valores não sofreram variações que pudessem desconsiderar os dados coletados. Outra conclusão

que pode ser observada a partir dos resultados é que a média dos valores obtidos é bem próxima aos valores coletados, pois estes não sofreram grandes variações.

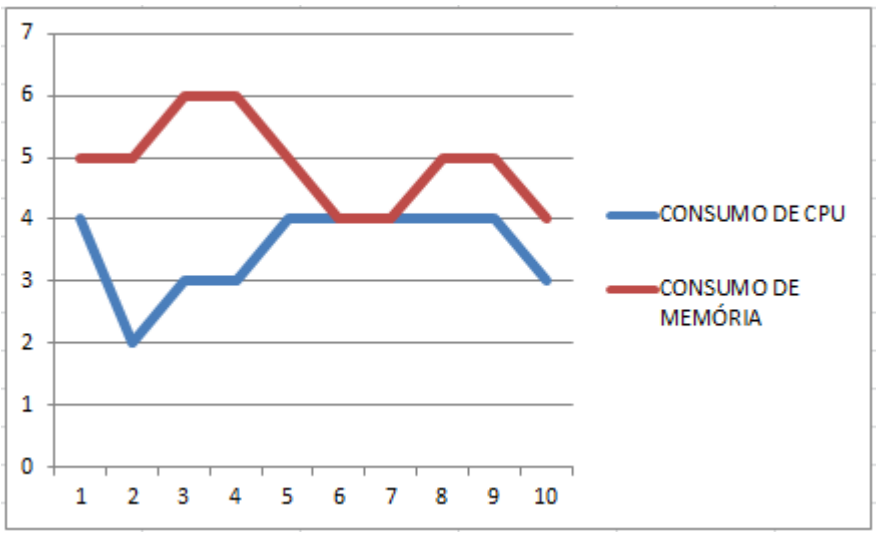
A Figura 19 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas antes da implentação da técnica de programação T3: Usar Sintaxe Aprimorada da Estrutura de Repetição for.

Figura 19 – Usar sintaxe aprimorada da estrutura de repetição for antes da implementação



A Figura 20 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas após da implentação da técnica de programação T3: Usar sintaxe aprimorada da estrutura de repetição for.

Figura 20 – Usar sintaxe aprimorada da estrutura de repetição for após a implementação



Analisando a métrica correspondente ao consumo de memória na Figura 19 observou-se uma oscilação de 0,01 para baixo ou para cima em relação à moda da medição. Na Figura 20 verificou-se que essa oscilação ocorreu apenas 0,01 para baixo ou para cima do valor da moda.

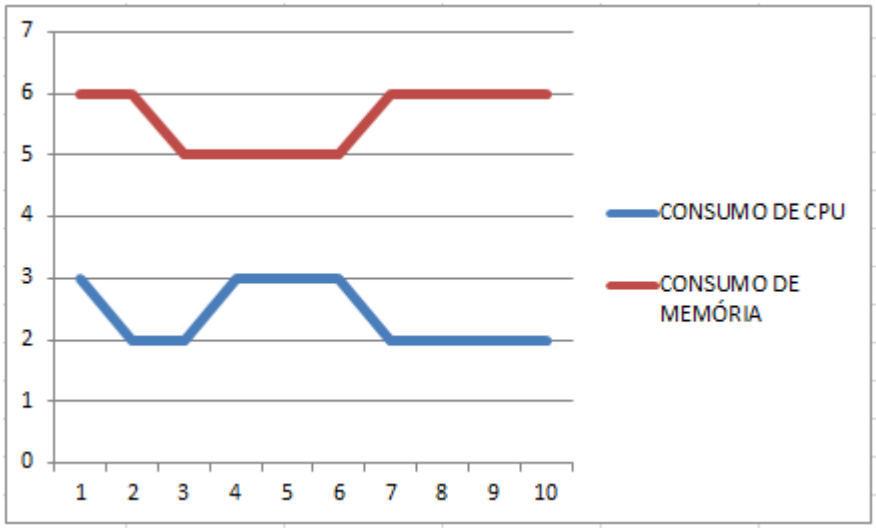
Analisando a métrica correspondente ao consumo de CPU nas Figuras 19 observou-se uma oscilação de 0,01 para baixo em relação à moda da medição. Na Figura 20 verificou-se que essa oscilação ocorreu de até 0,02 para baixo do valor da moda.

Com base nas análises de cada métrica nas Figuras 19 e 20 verificou-se que o valor

da última casa decimal é repetida durante vários testes o que confirma que os valores não sofreram variações que pudessem desconsiderar os dados coletados. Outra conclusão que pode ser observada a partir dos resultados é que a média dos valores obtidos é bem próxima aos valores coletados, pois estes não sofreram grandes variações.

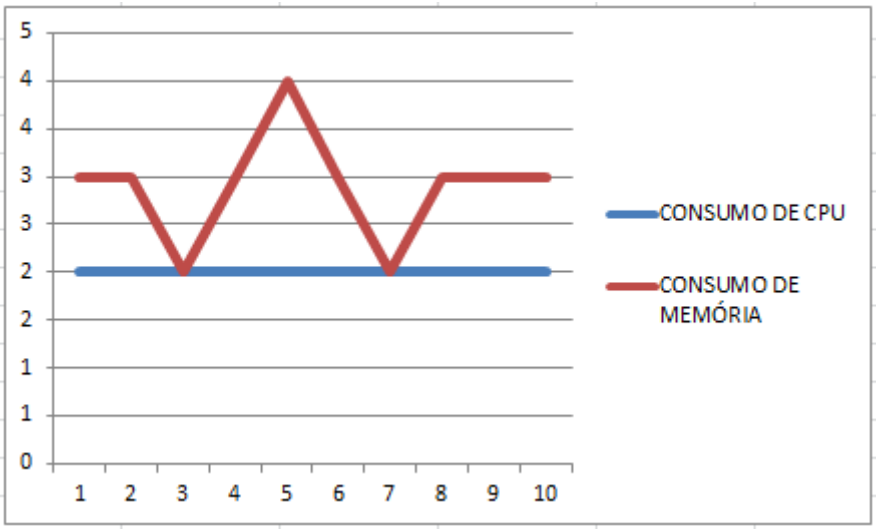
A Figura 21 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas antes da implentação da técnica de programação T4: Evitar Uso de Ponto Flutuante.

Figura 21 – Evitar uso de ponto flutuante antes da implementação



A Figura 22 apresenta a variação da última casa decimal dos dez valores medidos para cada uma das duas métricas após da implentação da técnica de programação T4: Evitar Uso de Ponto Flutuante.

Figura 22 – Evitar uso de ponto flutuante após a implementação



Analisando a métrica correspondente ao consumo de memória na Figura 21 observou-se uma oscilação de 0,01 para baixo em relação à moda da medição. Na Figura 22 verificou-

se que essa oscilação ocorreu apenas 0,01 para baixo ou para cima do valor da moda.

Analisando a métrica correspondente ao consumo de CPU nas Figuras 21 observou-se uma oscilação de 0,01 para cima em relação à moda da medição. Na Figura 22 não verificou-se oscilação dos valores.

Com base nas análises de cada métrica nas Figuras 21 e 22 verificou-se que o valor da última casa decimal é repetida durante vários testes o que confirma que os valores não sofreram variações que pudessem desconsiderar os dados coletados. Outra conclusão que pode ser observada a partir dos resultados é que a média dos valores obtidos é bem próxima aos valores coletados, pois estes não sofreram grandes variações.

6.4.3 Análise Quantitativa

Como foi citado no início desse capítulo, o objetivo deste trabalho é avaliar o impacto de um conjunto de técnicas de programação no desempenho de um aplicativo desenvolvido para Android, a fim de verificar qual a melhor em relação a um determinado recurso desses dispositivos.

As Tabelas 11 e 12 mostra a média antes e após a implementação das técnicas de programação, o impacto real da técnica de programação em cada recurso do dispositivo móvel.

Para entender a Tabela 11 utilize a legenda abaixo:

- T1: Utilizar estático ao invés de virtual
- T2: Evite Getters/Setters internos
- T3: Usar sintaxe aprimorada da estrutura de repetição for
- T4: Evitar uso de ponto flutuante

Tabela 11 – Impacto real com relação ao Consumo de Memória

TECNICAS DE PROGRAMAÇÃO	M1 - Consumo de Memória		
	ANTES	APÓS	VALOR DO IMPACTO
T1	34.551	34.253	-0.298
T2	34.565	34.618	+0.053
T3	34.559	34.149	-0.410
T4	34.556	34.329	-0.227

Como pode ser visto na Tabela 11, há indícios de que somente a técnicas de programação: T1-Utilizar Estático ao Invés de Virtual, afetou de maneira negativa o

consumo de mémoria do dispositivo móvel e que a melhor técnica de programação para ser aplicada no aplicativos desenvolvidos para os dispositivos móveis que utilizam o sistema operacional android é a: T3-Usar Sintaxe Aprimorada da Estrutura de Repetição for.

Tabela 12 – Impacto real com relação ao Consumo de CPU

TECNICAS DE PROGRAMAÇÃO	M2 - Consumo de CPU		
	ANTES	APÓS	VALOR DO IMPACTO
T1	22.7	22.7	0
T2	23	22	-1
T3	22.7	24	+1.3
T4	22.4	22	-0.4

Como pode ser visto na Tabela 12, há indícios de que as técnicas de programação: T2- Evite Getters/Setters internos e T4-Evitar Uso de Ponto Flutuante afetaram de maneira positiva o consumo de CPU do dispositivo móvel; a técnica de programação: T1-Utilizar Estático ao Invés de Virtual não afetou o consumo de CPU do dispositivo móvel; a técnica de programação: T3- Usar Sintaxe Aprimorada da Estrutura de Repetição for afetou de maneira negativa o consumo de CPU do dispositivo móvel; e que a melhor técnica de programação para ser aplicada no aplicativos desenvolvidos para os dispositivos móveis que utilizam o sistema operacional Android é a: T2- Evite Getters/Setters internos.

O resultado deste experimento mostrou que existem indícios de que a melhor técnica de programação para ser utilizada em um aplicativo cujo o foco seja obter um melhor desempenho com relação ao recurso **CPU** é a T2- Evite Getters/Setters internos. Caso o foco do aplicativo seja obter um melhor desempenho com relação à **memória**, a melhor técnica de programação é a T3- Usar sintaxe aprimorada da estrutura de repetição for.

Outro fato encontrado é que as técnicas de programação T1- Utilizar estático ao invés de virtual e T4- Evitar uso de ponto flutuante podem ser utilizadas pelos desenvolvedores não importando se o foco é um melhor desempenho de memória ou CPU, pois o seu uso afeta de maneira positiva ou não interfere no desempenho desses recursos.

6.4.4 Conclusão sobre as hipóteses

A análise dos dados coletados mostrou que há indícios de que o uso de algumas técnicas de programação provocam um ganho sobre o desempenho em relação aos recursos dos dispositivos móveis que utilizam o sistema operacional Android. No que diz respeito a consumo de CPU, a técnica de programação que obteve maior ganho em relação as demais foi: T2- Evite Getters/Setters internos. Com relação a consumo de mémoria a técnica de programação que obteve maior ganho em relação as demais foi: T3- Usar

sintaxe aprimorada da estrutura de repetição for. Desta forma, conclui-se que "Pelo menos uma das técnicas de programação *afeta positivamente* a utilização do recurso CPU e/ou memória dos dispositivos móveis." (Hípotese alternativa H1).

Outro indício encontrado é que a técnica de programação T1-Utilizar Estático ao Invés de Virtual não afetou o consumo de CPU do dispositivo analisado. Desta forma, conclui-se que "Pelo menos uma das técnicas de programação *não afeta* a utilização do recurso CPU e/ou memória dos dispositivos móveis." (Hípotese alternativa H0).

Analisando a ultima hipótese desse experimento, podemos observar nas considerações feitas anteriormente que as técnicas de programação: T2- Evite Getters/Setters internos e T3- Usar sintaxe aprimorada da estrutura de repetição for afetaram de maneira negativa respectivamente o consumo de memória e CPU. Desta forma, conclui-se que "Pelo menos uma das técnicas de programação *afeta negativamente* a utilização do recurso CPU e/ou memória dos dispositivos móveis." (Hípotese alternativa H2).

7 Considerações Finais

Este capítulo apresenta as considerações finais deste trabalho de conclusão de curso e a proposta de trabalhos futuros.

7.1 Conclusão

Motivado pela necessidade de aplicativos cada vez mais eficientes, pelo crescente número de dispositivos móveis utilizando o sistema operacional Android e pela quantidade limitada de recursos desses dispositivos, o objetivo deste trabalho é avaliar o impacto de um conjunto de técnicas de programação no desempenho de um aplicativo desenvolvido para Android, a fim de encontrar indícios de qual a melhor técnica de programação a um recurso desses dispositivos, visando um melhor desempenho.

Após revisão de literatura, foram definidas as técnicas de programação; as métricas de software que foram utilizadas para medir o desempenho do software; as ferramentas e outros itens do experimento executado no contexto dessa pesquisa a fim de verificar qual a melhor em relação aos recursos CPU e memória desses dispositivos móveis.

O resultado deste experimento mostrou que existem indícios de que a melhor técnica de programação para ser utilizada em um aplicativo cujo o foco seja obter um melhor desempenho com relação ao recurso **CPU** é a T2-Evite Getters/Setters Internos. Caso o foco do aplicativo seja obter um melhor desempenho com relação à **memória**, a melhor técnica de programação é a T3- Usar Sintaxe Aprimorada da Estrutura de Repetição for.

Outro fato encontrado é que as técnicas de programação T1-Utilizar Estático ao Invés de Virtual e T4-Evitar Uso de Ponto Flutuante podem ser utilizadas pelos desenvolvedores não importando se o foco é um melhor desempenho de memória ou CPU, pois o seu uso afeta de maneira positiva ou não interfere no desempenho desses recursos.

No entanto, os valores coletados neste experimento não podem ser comparados com os valores citados pela Google, pois ela não define o que é desempenho nem como é calculado o valor de impacto de cada técnica de programação com relação a desempenho.

7.2 Trabalhos Futuros

Existem várias técnicas de programação além das citadas neste trabalho. Desta forma, sugere-se como trabalho futuro, que sejam procuradas outras técnicas de programação para serem analisadas junto com o conjunto definido neste trabalho. Assim, será

possível verificar se existe outra técnica de programação que o impacto seja maior do que as técnicas analisadas neste trabalho.

Além disso, foram analisadas apenas dois aspectos do desempenho: CPU e memória, de acordo com os resultados obtidos no survey conduzido neste trabalho. Assim, sugere-se como trabalho futuro que seja feita uma análise em relação a outros recursos, por exemplo, utilização de bateria em dispositivos móveis.

Em relação às técnicas de programação analisadas neste trabalho, sugere-se como trabalho futuro um estudo de como o conjunto dessas técnicas interferem no desempenho das aplicações em dispositivos móveis.

Em relação à comparação entre os valores obtidos neste experimento e citados pela Google, sugere-se como trabalho futuro um estudo sobre uma fórmula para cálculo de desempenho que viabilize essa comparação.

Em relação à fadiga causada pela repetição da operação do experimento deste trabalho, sugere-se como trabalho futuro uma automatização do processo de operação que neste experimento é realizada de maneira totalmente manual.

Referências

ANATEL. 2015. <http://www.anatel.gov.br/Portal/exibirPortalInternet.do>. Acessado em 20/05/2015. Citado na página 19.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. *College Park. University of Maryland*, 1994. Citado 3 vezes nas páginas 13, 41 e 42.

BERGHOUT, E.; SOLINGEN, R. V. *THE GOAL/QUESTION/METRIC METHOD: a practical guide for quality improvement of software development*. [S.l.]: Company, 1999. Citado na página 41.

BRERETON, P. et al. Lessons from applying the systematic literature review process within the software engineering domain. *5th International Workshop on Software and Performance*, 2006. Citado 2 vezes nas páginas 13 e 35.

BRITO, V. M. de. Proposta de um conjunto de competências para um product owner. 2014. Citado na página 26.

DEVELOPER, G. *Developer Tools*. 2013. <http://developer.android.com/tools/index.html>. Acessado em 24/04/2014. Citado 2 vezes nas páginas 38 e 39.

ECLIPSE. 2013. <http://www.eclipse.org>. Acessado em 24/04/2014. Citado na página 39.

ENGHOLM, H. *Engenharia de Software na Prática*. 2010. <http://novatec.com.br/livros/engenhariasoftware/capitulo9788575222171.pdf>. Acesso em: 23/03/2014. Citado na página 21.

FINK, A. *How to Conduct Surveys*. [S.l.: s.n.], 2003. Citado na página 26.

FREITAS, H. et al. O metodo de pesquis survey. 2000. Citado na página 26.

FSF. *Free Software Foundation*. 2004. <https://www.fsf.org/pt-br>. Acessado em 30/04/2014. Citado 2 vezes nas páginas 29 e 40.

GOOGLE. *Performance Tips*. 2013. <http://developer.android.com/training/articles/perf-tips.html>. Acessado em 10/03/2014. Citado 2 vezes nas páginas 25 e 30.

GUNTHER, H. Como elaborar um questionário. 2003. Citado na página 26.

HUB, G. *Git Hub*. 2014. <https://github.com/>. Acessado em 01/05/2014. Citado na página 40.

ISO/IEC. *Systems and software engineering – Measurement process*. 2007. Citado 3 vezes nas páginas 13, 42 e 43.

MCGARRY, J. et al. *Practical Software Measurement: Objective Information for Decision MakersHardcover*. [S.l.]: ciencia, 2001. Citado 4 vezes nas páginas 15, 43, 44 e 45.

OHA. *Open Handset Alliance*. 2009. <http://www.openhandsetalliance.com/oha_overview.html>. Acessado em 20/03/2014. Citado na página 19.

PEREIRA, L. C. O.; SILVA, M. L. da. *Android para Desenvolvedores*. [S.l.]: BRASPORT, 2009. Citado 4 vezes nas páginas 19, 20, 21 e 25.

RAMOS, C. S. *Notas de Aula da Disciplina Medição e Análise*. 2013. Disciplina ministrada no primeiro semestre de 2013. Citado na página 50.

RANGARAJAN, K. et al. *Little Eye Labs*. 2013. <http://www.littleeye.co/>. Acessado em 25/04/2014. Citado 2 vezes nas páginas 39 e 40.

SILVA, E. L. da; MENEZES, E. M. *Metodologia da Pesquisa e Elaboração de Dissertação*. [S.l.]: Florianópolis, 2001. Citado na página 21.

SILVA, R. M. da. Qualidade na modelagem de processos de software. 2014. Citado na página 26.

WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Software Engineering, 2012. Citado 2 vezes nas páginas 47 e 50.

YAMIN, A. et al. Explorando o escalonamento no desempenho de aplicações móveis distribuídas. *ciencia*, 2001. Citado 2 vezes nas páginas 20 e 25.

Apêndices

APÊNDICE A – Dados completos das medições

Para entender as tabelas utilize a legenda abaixo:

- T1: Utilizar estático ao invés de virtual
- T2: Evite Getters/Setters internos
- T3: Usar sintaxe aprimorada da estrutura de repetição for
- T4: Evitar uso de ponto flutuante

A Tabela 13 apresenta os dados coletados, em megabyte, referentes a métrica consumo de métrica antes e após a implementação das quatro técnicas de programação selecionadas.

Tabela 13 – Métrica: Consumo de Memória

TECNICAS DE PROGRAMAÇÃO	M2 - Consumo de Memória	
	ANTES DA IMPLEMENTAÇÃO	APÓS A IMPLEMENTAÇÃO
T1	34.55 34.56 34.55 34.55 34.55 34.54 34.56 34.55 34.55 34.55	34.25 34.26 34.25 34.25 34.25 34.26 34.26 34.25 34.25 34.25
T2	34.57 34.56 34.55 34.57 34.57 34.57 34.56 34.56 34.57 34.57	34.63 34.60 34.62 34.63 34.60 34.60 34.62 34.62 34.63 34.63
T3	34.56 34.56 34.55 34.55 34.56 34.55 34.56 34.56 34.57 34.57	34.15 34.15 34.16 34.16 34.15 34.14 34.14 34.15 34.15 34.14
T4	34.56 34.56 34.55 34.55 34.55 34.55 34.56 34.56 34.56 34.56	34.33 34.33 34.32 34.33 34.34 34.33 34.32 34.33 34.33 34.33

A Tabela 14 apresenta os dados coletados, em procentagem, referentes a métrica consumo de CPU antes e após a implementação das quatro técnicas de programação selecionadas.

Tabela 14 – Métrica: Consumo de CPU

TECNICAS DE PROGRAMAÇÃO	M2 - Consumo de CPU	
	ANTES DA IMPLEMENTAÇÃO	APÓS A IMPLEMENTAÇÃO
T1	23 22 22 23 23 23 22 23 23 23	23 22 22 23 23 23 22 23 23 23
T2	23 23 23 23 23 23 23 23 23 23	23 22 22 22 23 23 22 22 22 23
T3	23 22 23 23 23 22 23 23 22 23	24 22 23 23 24 24 24 24 24 23
T4	23 22 22 23 23 23 22 22 22 22	22 22 22 22 22 22 22 22 22 22

APÊNDICE B – Códigos antes da implentação

As três classes a seguir mostram as classes do aplicativo CPU spy *antes das alterações* feitas para implentar cada técnica de programação:

```
1  //
       ----------------------------------------------------------------------------
2  //
3  // (C) Brandon Valosek, 2011 <bvalosek@gmail.com>
4  // (A) Victor Hugo Alves de Carvalho, 2015 <victorhugodf.ac@gmail.com>
5  //
       ----------------------------------------------------------------------------
6
7  package com.bvalosek.cpuspy4;
8
9  // imports
10 import java.io.BufferedReader;
11 import java.io.FileInputStream;
12 import java.io.IOException;
13 import java.io.InputStream;
14 import java.io.InputStreamReader;
15 import java.util.HashMap;
16 import java.util.Iterator;
17 import java.util.Map;
18 import android.app.Application;
19 import android.content.SharedPreferences;
20 import android.util.Log;
21 import com.bvalosek.cpuspy.CpuStateMonitor.CpuState;
22 import com.bvalosek.cpuspy.CpuStateMonitor.CpuStateMonitorException;
23
24 public class CpuSpyApp extends Application {
25
26   public  CpuStateMonitor _monitor = new CpuStateMonitor();
27
28     private String KERNEL_VERSION_PATH = "/proc/version";
29
30     private String TAG = "CpuSpyApp";
31
32     private String PREF_NAME = "CpuSpyPreferences";
33     private String PREF_OFFSETS = "offsets";
```

```
34
35      private String _kernelVersion = "";
36
37      /**
38       * On application start, load the saved offsets and stash the
39       * current kernel version string
40       */
41      @Override public void onCreate(){
42          loadOffsets();
43          updateKernelVersion();
44      }
45
46      /**
47       * Load the saved string of offsets from preferences and put it into
48       * the state monitor
49       */
50      public void loadOffsets() {
51          SharedPreferences settings = getSharedPreferences(
52                  PREF_NAME, MODE_PRIVATE);
53          String prefs = settings.getString (PREF_OFFSETS, "");
54
55          if (prefs == null || prefs.length() < 1) {
56              return;
57          }
58
59          // split the string by peroids and then the info by commas and load
60          Map<Integer, Long> offsets = new HashMap<Integer, Long>();
61          String[] sOffsets = prefs.split(",");
62          for(int i=0; i<sOffsets.length; i++) {
63            String[] parts = sOffsets[i].split(" ");
64              offsets.put (Integer.parseInt(parts[0]),
65                           Long.parseLong(parts[1]));
66          }
67
68          _monitor.getOffsets = offsets;
69      }
70
71      /**
72       * Save the state-time offsets as a string
73       * e.g. "100 24, 200 251, 500 124 etc
74       */
75      public void saveOffsets() {
76          SharedPreferences settings = getSharedPreferences(
77                  PREF_NAME, MODE_PRIVATE);
78          SharedPreferences.Editor editor = settings.edit();
79
80          // build the string by iterating over the freq->duration map
```

```
81          String str = "";
82          Iterator<Map.Entry<Integer, Long>> iterator =
83          _monitor._offsets.entrySet().iterator();
84          while(iterator.hasNext()) {
85            Map.Entry<Integer, Long> entry = iterator.next();
86            str += entry.getKey() + " " + entry.getValue() + ",";
87          }
88
89          editor.putString(PREF_OFFSETS, str);
90          editor.commit();
91      }
92
93      /** Try to read the kernel version string from the proc fileystem */
94      public String updateKernelVersion() {
95          try {
96              InputStream is = new FileInputStream(KERNEL_VERSION_PATH);
97              InputStreamReader ir = new InputStreamReader(is);
98              BufferedReader br = new BufferedReader(ir);
99
100             String line;
101             while ((line = br.readLine())!= null ) {
102                 setOffsets(line);
103             }
104
105             is.close();
106         } catch (IOException e) {
107             Log.e(TAG, "Problem reading kernel version file");
108             return "";
109         }
110
111         // made it
112         return getOffsets;
113     }
114
115
116     public String getKernelVersion() {
117         return _kernelVersion;
118     }
119
120     public void setKernelVersion(String kernelVersion) {
121         _kernelVersion = kernelVersion;
122     }
123
124 }
```

```
1  //
   ------------------------------------------------------------------------------
2  //
3  // (C) Brandon Valosek, 2011 <bvalosek@gmail.com>
4  // (A) Victor Hugo Alves de Carvalho, 2015 <victorhugodf.ac@gmail.com>
5  //
   ------------------------------------------------------------------------------
6
7  package com.bvalosek.cpuspy4;
8
9  // imports
10 import java.io.BufferedReader;
11 import java.io.FileInputStream;
12 import java.io.IOException;
13 import java.io.InputStream;
14 import java.io.InputStreamReader;
15 import java.util.ArrayList;
16 import java.util.Collections;
17 import java.util.HashMap;
18 import java.util.Iterator;
19 import java.util.List;
20 import java.util.Map;
21 import java.util.Set;
22 import android.os.SystemClock;
23
24 public class CpuStateMonitor {
25
26     public String TIME_IN_STATE_PATH =
27         "/sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state";
28
29     private String TAG = "CpuStateMonitor";
30
31     private List<CpuState>       _states = new ArrayList<CpuState>();
32     private Map<Integer, Long>   _offsets = new HashMap<Integer, Long>();
33
34     /** exception class */
35     public class CpuStateMonitorException extends Exception {
36         public CpuStateMonitorException(String s) {
37             super(s);
38         }
39     }
40
41     /**
42      * simple struct for states/time
43      */
```

```
44      public class CpuState implements Comparable<CpuState> {
45          /** init with freq and duration */
46          public CpuState(float a, long b) { freq = a; duration = b; }
47
48          public float freq = 0;
49          public long duration = 0;
50
51          /** for sorting, compare the freqs */
52          public float compareTo(CpuState state) {
53              Integer a = new Integer(freq);
54              Integer b = new Integer(state.freq);
55              return a.compareTo(b);
56          }
57      }
58
59      /** @return List of CpuState with the offsets applied */
60      public List<CpuState> getStates() {
61          List<CpuState> states = new ArrayList<CpuState>();
62
63          /* check for an existing offset, and if it's not too big,
64           * subtract it from the duration, otherwise just add it
65       * to the return List */
66          for(int i=0; i<getStates.size(); i++) {
67              long duration = getStates.get(i).duration;
68              if (getOffsets.containsKey(getStates.get(i).freq)) {
69                  long offset = getOffsets.get(getStates.get(i).freq);
70                  if (offset <= duration) {
71                      duration -= offset;
72                  } else {
73                      /* offset > duration implies our offsets are
74                       * now invalid, so clear and recall this function */
75                      getOffsets.clear();
76                      return getStates();
77                  }
78              }
79
80              states.add(new CpuState(getStates.get(i).freq, duration));
81          }
82
83          return states;
84      }
85
86      /**
87       * @return Sum of all state durations including deep sleep, accounting
88       * for offsets
89       */
90      public long getTotalStateTime() {
```

```
91          long sum = 0;
92          long offset = 0;
93
94          for(int i=0; i<getStates.size(); i++) {
95              sum += getStates.get(i).duration;
96          }
97
98          Iterator<Map.Entry<Integer, Long>> iterator =
99          getOffsets.entrySet().iterator();
100         while(iterator.hasNext()) {
101             offset += iterator.next().getValue();
102         }
103
104         return sum - offset;
105     }
106
107     /**
108      * Updates the current time in states and then sets the offset map
109      * to the current duration, effectively "zeroing out" the timers
110      */
111     public void setOffsets() throws CpuStateMonitorException {
112         getOffsets.clear();
113         updateStates();
114
115         for(int i=0; i<getStates.size(); i++) {
116             getOffsets.put(getStates.get(i).freq,
117         getStates.get(i).duration);
118         }
119     }
120
121     /** removes state offsets */
122     public void removeOffsets() {
123         getOffsets.clear();
124     }
125
126     /**
127      * @return a list of all the CPU frequency states, which contains
128      * both a frequency and a duration (time spent in that state
129      */
130     public List<CpuState> updateStates()
131         throws CpuStateMonitorException {
132         /* attempt to create a buffered reader to the time in state
133          * file and read in the states to the class */
134         try {
135             InputStream is = new FileInputStream(TIME_IN_STATE_PATH);
136             InputStreamReader ir = new InputStreamReader(is);
137             BufferedReader br = new BufferedReader(ir);
```

```
138             getStates.clear();
139             readInStates(br);
140             is.close();
141         } catch (IOException e) {
142             throw new CpuStateMonitorException(
143                     "Problem opening time-in-states file");
144         }
145 
146         /* deep sleep time determined by difference between elapsed
147          * (total) boot time and the system uptime (awake) */
148         long sleepTime = (SystemClock.elapsedRealtime()
149                 - SystemClock.uptimeMillis()) / 10;
150         getStates.add(new CpuState(0, sleepTime));
151 
152         Collections.sort(getStates, Collections.reverseOrder());
153 
154         return getStates;
155     }
156 
157     /** read from a provided BufferedReader the state lines into the
158      * States member field
159      */
160     private void readInStates(BufferedReader br)
161         throws CpuStateMonitorException {
162         try {
163             String line;
164             while ((line = br.readLine()) != null) {
165                 // split open line and convert to Integers
166                 String[] nums = line.split(" ");
167                 getStates.add(new CpuState(
168                         Integer.parseInt(nums[0]),
169                         Long.parseLong(nums[1])));
170             }
171         } catch (IOException e) {
172             throw new CpuStateMonitorException(
173                     "Problem processing time-in-states file");
174         }
175     }
176 
177   public List<CpuState> getStates() {
178         return _states;
179     }
180 
181       public void setStates(List<CpuState> states) {
182         _states = states;
183     }
184 
```

```
185
186     public Map<Integer, Long> getOffsets() {
187         return _offsets;
188     }
189
190     /** Sets the offset map (freq->duration offset) */
191     public void setOffsets(Map<Integer, Long> offsets) {
192         _offsets = offsets;
193     }
194
195 }
```

```
1  //
   ------------------------------------------------------------------------------

2  //
3  // (C) Brandon Valosek, 2011 <bvalosek@gmail.com>
4  // (A) Victor Hugo Alves de Carvalho, 2015 <victorhugodf.ac@gmail.com>
5  //
   ------------------------------------------------------------------------------

6
7  package com.bvalosek.cpuspy4.ui;
8
9  // imports
10 import java.util.ArrayList;
11 import java.util.List;
12
13 import android.app.Activity;
14 import android.content.Context;
15 import android.os.AsyncTask;
16 import android.os.Bundle;
17 import android.os.Debug;
18 import android.view.LayoutInflater;
19 import android.view.Menu;
20 import android.view.MenuInflater;
21 import android.view.MenuItem;
22 import android.view.View;
23 import android.view.ViewGroup;
24 import android.widget.LinearLayout;
25 import android.widget.ProgressBar;
26 import android.widget.TextView;
27 import com.bvalosek.cpuspy.*;
28 import com.bvalosek.cpuspy.CpuStateMonitor.CpuState;
29 import com.bvalosek.cpuspy.CpuStateMonitor.CpuStateMonitorException;
30 import android.util.Log;
31
32 public class HomeActivity extends Activity
33 {
34     private String TAG = "CpuSpy";
35
36     private CpuSpyApp _app = null;
37
38     // the views
39     private LinearLayout     _uiStatesView = null;
40     private TextView         _uiAdditionalStates = null;
41     private TextView         _uiTotalStateTime = null;
42     private TextView         _uiHeaderAdditionalStates = null;
43     private TextView         _uiHeaderTotalStateTime = null;
```

```
44      private TextView        _uiStatesWarning = null;
45      private TextView        _uiKernelString = null;
46
47      /** whether or not we're updating the data in the background */
48      private boolean     _updatingData = false;
49
50      /** Initialize the Activity */
51      @Override
52      public void onCreate(Bundle savedInstanceState)
53      {
54          super.onCreate(savedInstanceState);
55
56          // inflate the view, stash the app context, and get all UI elements
57          setContentView(R.layout.home_layout);
58          _app = (CpuSpyApp)getApplicationContext();
59          findViews();
60
61          // set title to version string
62          setTitle("CpuSpy");
63
64          if (savedInstanceState != null) {
65              setUpdatingData(savedInstanceState.getBoolean("updatingData"));
66          }
67      }
68
69      /** When the activity is about to change orientation */
70      @Override
71      public void onSaveInstanceState(Bundle outState) {
72          super.onSaveInstanceState(outState);
73          outState.putBoolean("updatingData", getUpdatingData);
74      }
75
76
77      /** Update the view when the application regains focus */
78      @Override
79      public void onResume () {
80          super.onResume();
81          refreshData();
82      }
83
84      /** Map all of the UI elements to member variables */
85      private void findViews() {
86          _uiStatesView = (LinearLayout)findViewById(
87        R.id.ui_states_view);
88          _uiKernelString = (TextView)findViewById(
89        R.id.ui_kernel_string);
90          _uiAdditionalStates = (TextView)findViewById(
```

```
91          R.id.ui_additional_states);
92              _uiHeaderAdditionalStates = (TextView)findViewById(
93          R.id.ui_header_additional_states);
94              _uiHeaderTotalStateTime = (TextView)findViewById(
95          R.id.ui_header_total_state_time);
96              _uiStatesWarning = (TextView)findViewById(
97          R.id.ui_states_warning);
98              _uiTotalStateTime = (TextView)findViewById(
99          R.id.ui_total_state_time);
100     }
101
102     /** called when we want to infalte the menu */
103     @Override
104     public boolean onCreateOptionsMenu(Menu menu) {
105         // request inflater from activity and inflate into its menu
106         MenuInflater inflater = getMenuInflater();
107         inflater.inflate(R.menu.home_menu, menu);
108
109         // made it
110         return true;
111     }
112
113     /** called to handle a menu event */
114     @Override
115     public boolean onOptionsItemSelected(MenuItem item) {
116         // what it do mayne
117         switch (item.getItemId()) {
118         /* pressed the load menu button */
119         case R.id.menu_refresh:
120             refreshData();
121             break;
122         case R.id.menu_reset:
123             try {
124                 _app._monitor.setOffsets();
125             } catch (CpuStateMonitorException e) {
126                 // TODO: something
127             }
128
129             _app.saveOffsets();
130             updateView();
131             break;
132         case R.id.menu_restore:
133             _app._monitor.removeOffsets();
134             _app.saveOffsets();
135             updateView();
136             break;
137         }
```

```
138
139         // made it
140         return true;
141     }
142
143     /** Generate and update all UI elements */
144     public void updateView() {
145       Debug.startMethodTracing("CpuSpy");
146         /** Get the CpuStateMonitor from the app, and iterate over
147          * all states,creating a row if the duration is > 0 or otherwise
148          * marking it in extraStates (missing) */
149         CpuStateMonitor monitor = _app._monitor;
150         _uiStatesView.removeAllViews();
151         List<String> extraStates = new ArrayList<String>();
152         for(int i=0; i<monitor.getStates().size(); i++) {
153             if (monitor.getStates().get(i).duration > 0) {
154                 generateStateRow(
155         monitor.getStates().get(i),
156         _uiStatesView);
157             } else {
158                 if (monitor.getStates().get(i).freq == 0) {
159                     extraStates.add("Deep Sleep");
160                 } else {
161                     extraStates.add(monitor.getStates().get(i).freq/1000
162           + " MHz");
163                 }
164             }
165         }
166
167         // show the red warning label if no states found
168         if ( monitor.getStates().size() == 0) {
169             _uiStatesWarning.setVisibility(View.VISIBLE);
170             _uiHeaderTotalStateTime.setVisibility(View.GONE);
171             _uiTotalStateTime.setVisibility(View.GONE);
172             _uiStatesView.setVisibility(View.GONE);
173         }
174
175         // update the total state time
176         long totTime = monitor.getTotalStateTime() / 100;
177         _uiTotalStateTime.setText(sToString(totTime));
178
179         // for all the 0 duration states, add the the Unused State area
180         if (extraStates.size() > 0) {
181             int n = 0;
182             String str = "";
183
184             for(int i=0; i< extraStates.size(); i++) {
```

```
185                 if (n++ > 0)
186                     str += ", ";
187                 str += extraStates.get(i);
188             }
189 
190             _uiAdditionalStates.setVisibility(View.VISIBLE);
191             _uiHeaderAdditionalStates.setVisibility(View.VISIBLE);
192             _uiAdditionalStates.setText(str);
193         } else {
194             _uiAdditionalStates.setVisibility(View.GONE);
195             _uiHeaderAdditionalStates.setVisibility(View.GONE);
196         }
197 
198         // kernel line
199         _uiKernelString.setText(_app._kernelVersion);
200         Debug.stopMethodTracing();
201     }
202 
203     /** Attempt to update the time-in-state info */
204     public void refreshData() {
205         if (!getUpdatingData) {
206             new RefreshStateDataTask().execute((Void)null);
207         }
208     }
209 
210     /** @return A nicely formatted String representing tSec seconds */
211     private String sToString(long tSec) {
212         long h = (long)Math.floor(tSec / (60*60));
213         long m = (long)Math.floor((tSec - h*60*60) / 60);
214         long s = tSec % 60;
215         String sDur;
216         sDur = h + ":";
217         if (m < 10)
218             sDur += "0";
219         sDur += m + ":";
220         if (s < 10)
221             sDur += "0";
222         sDur += s;
223 
224         return sDur;
225     }
226 
227     /**
228      * @return a View that correpsonds to a CPU freq state
229      * row as specified by the state parameter
230      */
231     private View generateStateRow(CpuState state, ViewGroup parent) {
```

```
232         // inflate the XML into a view in the parent
233         LayoutInflater inf = LayoutInflater.from((Context)_app);
234         LinearLayout theRow = (LinearLayout)inf.inflate(
235                 R.layout.state_row, parent, false);
236 
237         // what percetnage we've got
238         CpuStateMonitor monitor = _app._monitor;
239         float per = (float)state.duration * 100 /
240             monitor.getTotalStateTime();
241         String sPer = (float)per + "%";
242 
243         // state name
244         String sFreq;
245         if (state.freq == 0) {
246             sFreq = "Deep Sleep";
247         } else {
248             sFreq = state.freq / 1000 + " MHz";
249         }
250 
251         // duration
252         long tSec = state.duration / 100;
253         String sDur = sToString(tSec);
254 
255         // map UI elements to objects
256         TextView freqText = (TextView)theRow.findViewById(
257       R.id.ui_freq_text);
258         TextView durText = (TextView)theRow.findViewById(
259                 R.id.ui_duration_text);
260         TextView perText = (TextView)theRow.findViewById(
261                 R.id.ui_percentage_text);
262         ProgressBar bar = (ProgressBar)theRow.findViewById(
263         R.id.ui_bar);
264 
265         // modify the row
266         freqText.setText(sFreq);
267         perText.setText(sPer);
268         durText.setText(sDur);
269         bar.setProgress((float)per);
270 
271         // add it to parent and return
272         parent.addView(theRow);
273         return theRow;
274     }
275 
276     /** Keep updating the state data off the UI thread for slow devices */
277     protected class RefreshStateDataTask extends AsyncTask<Void, Void, Void
    > {
```

```
278
279         /** Stuff to do on a seperate thread */
280         @Override protected Void doInBackground(Void... v) {
281             CpuStateMonitor monitor = _app._monitor;
282             try {
283                 monitor.updateStates();
284             } catch (CpuStateMonitorException e) {
285                 Log.e(TAG, "Problem getting CPU states");
286             }
287
288             return null;
289         }
290
291         /** Executed on the UI thread right before starting the task */
292         @Override protected void onPreExecute() {
293             log("starting data update");
294       setUpdatingData(true);
295         }
296
297         /** Executed on UI thread after task */
298         @Override protected void onPostExecute(Void v) {
299             log("finished data update");
300             setUpdatingData(false);
301             updateView();
302         }
303     }
304
305     /** logging */
306     private void log(String s) {
307         Log.d(TAG, s);
308     }
309
310   public boolean getUpdatingData() {
311         return _updatingData;
312     }
313
314     public void setUpdatingData(boolean updatingData) {
315         _updatingData = updatingData;
316     }
317 }
```

APÊNDICE C – Códigos após a implentação

As três classes a seguir mostram as classes do aplicativo CPU spy *após as alterações* feitas para implentar cada técnica de programação:

```
1  //
       ----------------------------------------------------------------------
2  //
3  // (C) Brandon Valosek, 2011 <bvalosek@gmail.com>
4  // (A) Victor Hugo Alves de Carvalho, 2015 <victorhugodf.ac@gmail.com>
5  //
       ----------------------------------------------------------------------
6
7  package com.bvalosek.cpuspy1;
8
9  // imports
10 import java.io.BufferedReader;
11 import java.io.FileInputStream;
12 import java.io.IOException;
13 import java.io.InputStream;
14 import java.io.InputStreamReader;
15 import java.util.HashMap;
16 import java.util.Map;
17 import android.app.Application;
18 import android.content.SharedPreferences;
19 import android.util.Log;
20 import com.bvalosek.cpuspy1.CpuStateMonitor.CpuState;
21 import com.bvalosek.cpuspy1.CpuStateMonitor.CpuStateMonitorException;
22
23 /** main application class */
24 public class CpuSpyApp extends Application {
25
26     private static final String KERNEL_VERSION_PATH = "/proc/version";
27
28     private static final String TAG = "CpuSpyApp";
29
30     private static final String PREF_NAME = "CpuSpyPreferences";
31     private static final String PREF_OFFSETS = "offsets";
32
```

```
33      /** the long-living object used to monitor the system frequency states
        */
34      private CpuStateMonitor _monitor = new CpuStateMonitor();
35
36      private String _kernelVersion = "";
37
38      /**
39       * On application start, load the saved offsets and stash the
40       * current kernel version string
41       */
42      @Override public void onCreate(){
43          loadOffsets();
44          updateKernelVersion();
45      }
46
47      /** @return the kernel version string */
48      public  String getKernelVersion() {
49          return _kernelVersion;
50      }
51
52      /** @return the internal CpuStateMonitor object */
53      public CpuStateMonitor getCpuStateMonitor() {
54          return _monitor;
55      }
56
57      /**
58       * Load the saved string of offsets from preferences and put it into
59       * the state monitor
60       */
61      public static void loadOffsets() {
62          SharedPreferences settings = getSharedPreferences(
63                  PREF_NAME, MODE_PRIVATE);
64          String prefs = settings.getString (PREF_OFFSETS, "");
65
66          if (prefs == null || prefs.length() < 1) {
67              return;
68          }
69
70          Map<Integer, Long> offsets = new HashMap<Integer, Long>();
71          String[] sOffsets = prefs.split(",");
72          for (String offset : sOffsets) {
73              String[] parts = offset.split(" ");
74              offsets.put (Integer.parseInt(parts[0]),
75                           Long.parseLong(parts[1]));
76          }
77
78          _monitor.setOffsets(offsets);
```

```
79      }
80
81      /**
82       * Save the state-time offsets as a string
83       * e.g. "100 24, 200 251, 500 124 etc
84       */
85      public static void saveOffsets() {
86          SharedPreferences settings = getSharedPreferences(
87                  PREF_NAME, MODE_PRIVATE);
88          SharedPreferences.Editor editor = settings.edit();
89
90          // build the string by iterating over the freq->duration map
91          String str = "";
92          for (Map.Entry<Integer, Long> entry :
93        _monitor.getOffsets().entrySet()) {
94              str += entry.getKey() + " " + entry.getValue() + ",";
95          }
96
97          editor.putString(PREF_OFFSETS, str);
98          editor.commit();
99      }
100
101     /** Try to read the kernel version string from the proc fileystem */
102     public static String updateKernelVersion() {
103         try {
104             InputStream is = new FileInputStream(KERNEL_VERSION_PATH);
105             InputStreamReader ir = new InputStreamReader(is);
106             BufferedReader br = new BufferedReader(ir);
107
108             String line;
109             while ((line = br.readLine())!= null ) {
110                 _kernelVersion = line;
111             }
112
113             is.close();
114         } catch (IOException e) {
115             Log.e(TAG, "Problem reading kernel version file");
116             return "";
117         }
118
119         // made it
120         return _kernelVersion;
121     }
122 }
```

```
1  //
       ----------------------------------------------------------------------------
2  //
3  // (C) Brandon Valosek, 2011 <bvalosek@gmail.com>
4  // (A) Victor Hugo Alves de Carvalho, 2015 <victorhugodf.ac@gmail.com>
5  //
       ----------------------------------------------------------------------------
6
7  package com.bvalosek.cpuspy1;
8
9  // imports
10 import java.io.BufferedReader;
11 import java.io.FileInputStream;
12 import java.io.IOException;
13 import java.io.InputStream;
14 import java.io.InputStreamReader;
15 import java.util.ArrayList;
16 import java.util.Collections;
17 import java.util.HashMap;
18 import java.util.List;
19 import java.util.Map;
20 import android.os.SystemClock;
21
22 public class CpuStateMonitor {
23
24     public static final String TIME_IN_STATE_PATH =
25         "/sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state";
26
27     private static final String TAG = "CpuStateMonitor";
28
29     private List<CpuState>      _states = new ArrayList<CpuState>();
30     public Map<Integer, Long>  _offsets = new HashMap<Integer, Long>();
31
32     /** exception class */
33     public class CpuStateMonitorException extends Exception {
34         public CpuStateMonitorException(String s) {
35             super(s);
36         }
37     }
38
39     /**
40      * simple struct for states/time
41      */
42     public class CpuState implements Comparable<CpuState> {
43         /** init with freq and duration */
```

```
44          public CpuState(int a, long b) { freq = a; duration = b; }
45
46          public int freq = 0;
47          public long duration = 0;
48
49          /** for sorting, compare the freqs */
50          public int compareTo(CpuState state) {
51              Integer a = new Integer(freq);
52              Integer b = new Integer(state.freq);
53              return a.compareTo(b);
54          }
55      }
56
57      /** @return List of CpuState with the offsets applied */
58      public static List<CpuState> getStates() {
59          List<CpuState> states = new ArrayList<CpuState>();
60
61          /* check for an existing offset, and if it's not too big,
62           * subtract it from the duration, otherwise just
63       add it to the return List */
64          for (CpuState state : _states) {
65              long duration = state.duration;
66              if (_offsets.containsKey(state.freq)) {
67                  long offset = _offsets.get(state.freq);
68                  if (offset <= duration) {
69                      duration -= offset;
70                  } else {
71                      /* offset > duration implies our offsets are
72                       * now invalid, so clear and recall this function */
73                      _offsets.clear();
74                      return getStates();
75                  }
76              }
77
78              states.add(new CpuState(state.freq, duration));
79          }
80
81          return states;
82      }
83
84      /**
85       * @return Sum of all state durations including deep sleep, accounting
86       * for offsets
87       */
88      public static long getTotalStateTime() {
89          long sum = 0;
90          long offset = 0;
```

```
91
92          for (CpuState state : _states) {
93              sum += state.duration;
94          }
95
96          for (Map.Entry<Integer, Long> entry : _offsets.entrySet()) {
97              offset += entry.getValue();
98          }
99
100         return sum - offset;
101     }
102
103     public List<CpuState> getStates() {
104         return _states;
105     }
106
107       public void setStates(List<CpuState> states) {
108         _states = states;
109     }
110
111
112     /**
113      * @return Map of freq->duration of all the offsets
114      */
115     public Map<Integer, Long> getOffsets() {
116         return _offsets;
117     }
118
119     /** Sets the offset map (freq->duration offset) */
120     public void setOffsets(Map<Integer, Long> offsets) {
121         _offsets = offsets;
122     }
123
124     /**
125      * Updates the current time in states and then sets the offset
126      * map to the current duration, effectively "zeroing out"
127      * the timers */
128     public void setOffsets() throws CpuStateMonitorException {
129         _offsets.clear();
130         updateStates();
131
132         for (CpuState state : _states) {
133             _offsets.put(state.freq, state.duration);
134         }
135     }
136
137     /** removes state offsets */
```

```
138     public void removeOffsets() {
139         _offsets.clear();
140     }
141 
142     /**
143      * @return a list of all the CPU frequency states, which contains
144      * both a frequency and a duration (time spent in that state
145      */
146     public static List<CpuState> updateStates()
147         throws CpuStateMonitorException {
148         /* attempt to create a buffered reader to the time in state
149          * file and read in the states to the class */
150         try {
151             InputStream is = new FileInputStream(TIME_IN_STATE_PATH);
152             InputStreamReader ir = new InputStreamReader(is);
153             BufferedReader br = new BufferedReader(ir);
154             _states.clear();
155             readInStates(br);
156             is.close();
157         } catch (IOException e) {
158             throw new CpuStateMonitorException(
159                     "Problem opening time-in-states file");
160         }
161 
162         /* deep sleep time determined by difference between elapsed
163          * (total) boot time and the system uptime (awake) */
164         long sleepTime = (SystemClock.elapsedRealtime()
165                 - SystemClock.uptimeMillis()) / 10;
166         _states.add(new CpuState(0, sleepTime));
167 
168         Collections.sort(_states, Collections.reverseOrder());
169 
170         return _states;
171     }
172 
173     /** read from a provided BufferedReader the state lines into the
174      * States member field
175      */
176     private void readInStates(BufferedReader br)
177         throws CpuStateMonitorException {
178         try {
179             String line;
180             while ((line = br.readLine()) != null) {
181                 // split open line and convert to Integers
182                 String[] nums = line.split(" ");
183                 _states.add(new CpuState(
184                         Integer.parseInt(nums[0]),
```

```
185                             Long.parseLong(nums[1])));
186                 }
187             } catch (IOException e) {
188                 throw new CpuStateMonitorException(
189                         "Problem processing time-in-states file");
190             }
191         }
192 }
```

```
1  //
   ------------------------------------------------------------------------------
2  //
3  // (C) Brandon Valosek, 2011 <bvalosek@gmail.com>
4  // (A) Victor Hugo Alves de Carvalho, 2015 <victorhugodf.ac@gmail.com>
5  //
   ------------------------------------------------------------------------------
6
7  package com.bvalosek.cpuspy1.ui;
8
9  // imports
10 import java.util.ArrayList;
11 import java.util.List;
12 import java.lang.Object;
13 import android.app.Activity;
14 import android.content.Context;
15 import android.os.AsyncTask;
16 import android.os.Bundle;
17 import android.os.Debug;
18 import android.view.LayoutInflater;
19 import android.view.Menu;
20 import android.view.MenuInflater;
21 import android.view.MenuItem;
22 import android.view.View;
23 import android.view.ViewGroup;
24 import android.widget.LinearLayout;
25 import android.widget.ProgressBar;
26 import android.widget.TextView;
27 import com.bvalosek.cpuspy1.*;
28 import com.bvalosek.cpuspy1.CpuStateMonitor.CpuState;
29 import com.bvalosek.cpuspy1.CpuStateMonitor.CpuStateMonitorException;
30
31 import android.util.Log;
32
33 /** main activity class */
34 public class HomeActivity extends Activity
35 {
36     private static final String TAG = "CpuSpy";
37
38     private CpuSpyApp _app = null;
39
40     // the views
41     private LinearLayout     _uiStatesView = null;
42     private TextView         _uiAdditionalStates = null;
43     private TextView         _uiTotalStateTime = null;
```

```
44      private TextView        _uiHeaderAdditionalStates = null;
45      private TextView        _uiHeaderTotalStateTime = null;
46      private TextView        _uiStatesWarning = null;
47      private TextView        _uiKernelString = null;
48
49      /** whether or not we're updating the data in the background */
50      private boolean      _updatingData = false;
51
52      /** Initialize the Activity */
53      @Override public void onCreate(Bundle savedInstanceState)
54      {
55          super.onCreate(savedInstanceState);
56
57          setContentView(R.layout.home_layout);
58          _app = (CpuSpyApp)getApplicationContext();
59          findViews();
60
61          // set title to version string
62          //setTitle(getResources().getText(R.string.app_name) + " v" +
63          //getResources().getText(R.string.version_name));
64          setTitle("CpuSpy");
65
66          if (savedInstanceState != null) {
67              _updatingData = savedInstanceState.
68          getBoolean("updatingData");
69          }
70      }
71
72      /** When the activity is about to change orientation */
73      @Override public void onSaveInstanceState(Bundle outState) {
74          super.onSaveInstanceState(outState);
75          outState.putBoolean("updatingData", _updatingData);
76      }
77
78
79      /** Update the view when the application regains focus */
80      @Override public void onResume () {
81          super.onResume();
82          refreshData();
83      }
84
85      /** Map all of the UI elements to member variables */
86      private void findViews() {
87          _uiStatesView = (LinearLayout)findViewById(
88          R.id.ui_states_view);
89          _uiKernelString = (TextView)findViewById(
90          R.id.ui_kernel_string);
```

```
91          _uiAdditionalStates = (TextView)findViewById(
92                  R.id.ui_additional_states);
93          _uiHeaderAdditionalStates = (TextView)findViewById(
94                  R.id.ui_header_additional_states);
95          _uiHeaderTotalStateTime = (TextView)findViewById(
96                  R.id.ui_header_total_state_time);
97          _uiStatesWarning = (TextView)findViewById(
98          R.id.ui_states_warning);
99          _uiTotalStateTime = (TextView)findViewById(
100         R.id.ui_total_state_time);
101     }
102
103     /** called when we want to infalte the menu */
104     @Override public boolean onCreateOptionsMenu(Menu menu) {
105         // request inflater from activity and inflate into its menu
106         MenuInflater inflater = getMenuInflater();
107         inflater.inflate(R.menu.home_menu, menu);
108
109         // made it
110         return true;
111     }
112
113     /** called to handle a menu event */
114     @Override public boolean onOptionsItemSelected(MenuItem item) {
115         // what it do mayne
116         switch (item.getItemId()) {
117         /* pressed the load menu button */
118         case R.id.menu_refresh:
119             refreshData();
120             break;
121         case R.id.menu_reset:
122             try {
123                 _app.getCpuStateMonitor().setOffsets();
124             } catch (CpuStateMonitorException e) {
125                 // TODO: something
126             }
127
128             _app.saveOffsets();
129             updateView();
130             break;
131         case R.id.menu_restore:
132             _app.getCpuStateMonitor().removeOffsets();
133             _app.saveOffsets();
134             updateView();
135             break;
136         }
137
```

```
138         // made it
139         return true;
140     }
141 
142     /** Generate and update all UI elements */
143 
144     public static void updateView() {
145       Debug.startMethodTracing("CpuSpy");
146         /** Get the CpuStateMonitor from the app, and iterate
147          * over all states, creating a row if the duration is > 0 or
148          * otherwise marking it in extraStates (missing) */
149         CpuStateMonitor monitor = _app.getCpuStateMonitor();
150         _uiStatesView.removeAllViews();
151         List<String> extraStates = new ArrayList<String>();
152         for (CpuState state : monitor.getStates()) {
153             if (state.duration > 0) {
154                 generateStateRow(state, _uiStatesView);
155             } else {
156                 if (state.freq == 0) {
157                     extraStates.add("Deep Sleep");
158                 } else {
159                     extraStates.add(state.freq/1000 + " MHz");
160                 }
161             }
162         }
163 
164         // show the red warning label if no states found
165         if ( monitor.getStates().size() == 0) {
166             _uiStatesWarning.setVisibility(View.VISIBLE);
167             _uiHeaderTotalStateTime.setVisibility(View.GONE);
168             _uiTotalStateTime.setVisibility(View.GONE);
169             _uiStatesView.setVisibility(View.GONE);
170         }
171 
172         // update the total state time
173         long totTime = monitor.getTotalStateTime() / 100;
174         _uiTotalStateTime.setText(sToString(totTime));
175 
176         // for all the 0 duration states, add the the Unused State area
177         if (extraStates.size() > 0) {
178             int n = 0;
179             String str = "";
180 
181             for (String s : extraStates) {
182                 if (n++ > 0)
183                     str += ", ";
184                 str += s;
```

```
185             }
186
187             _uiAdditionalStates.setVisibility(View.VISIBLE);
188             _uiHeaderAdditionalStates.setVisibility(View.VISIBLE);
189             _uiAdditionalStates.setText(str);
190         } else {
191             _uiAdditionalStates.setVisibility(View.GONE);
192             _uiHeaderAdditionalStates.setVisibility(View.GONE);
193         }
194
195         // kernel line
196         _uiKernelString.setText(_app.getKernelVersion());
197         Debug.stopMethodTracing();
198     }
199
200
201     /** Attempt to update the time-in-state info */
202     public static void refreshData() {
203         if (!_updatingData) {
204             new RefreshStateDataTask().execute((Void)null);
205         }
206     }
207
208     /** @return A nicely formatted String representing tSec seconds */
209     private static String sToString(long tSec) {
210         long h = (long)Math.floor(tSec / (60*60));
211         long m = (long)Math.floor((tSec - h*60*60) / 60);
212         long s = tSec % 60;
213         String sDur;
214         sDur = h + ":";
215         if (m < 10)
216             sDur += "0";
217         sDur += m + ":";
218         if (s < 10)
219             sDur += "0";
220         sDur += s;
221
222         return sDur;
223     }
224
225     /**
226      * @return a View that correpsonds to a CPU freq state
227      * row as specified by the state parameter
228      */
229     private View generateStateRow(CpuState state, ViewGroup parent) {
230         // inflate the XML into a view in the parent
231         LayoutInflater inf = LayoutInflater.from((Context)_app);
```

```
232         LinearLayout theRow = (LinearLayout)inf.inflate(
233                 R.layout.state_row, parent, false);
234 
235         // what percetnage we've got
236         CpuStateMonitor monitor = _app.getCpuStateMonitor();
237         int per = (int)state.duration * 100 /
238             monitor.getTotalStateTime();
239         String sPer = (int)per + "%";
240 
241         // state name
242         String sFreq;
243         if (state.freq == 0) {
244             sFreq = "Deep Sleep";
245         } else {
246             sFreq = state.freq / 1000 + " MHz";
247         }
248 
249         // duration
250         long tSec = state.duration / 100;
251         String sDur = sToString(tSec);
252 
253         // map UI elements to objects
254         TextView freqText = (TextView)theRow.findViewById(R.id.ui_freq_text
    );
255         TextView durText = (TextView)theRow.findViewById(
256                 R.id.ui_duration_text);
257         TextView perText = (TextView)theRow.findViewById(
258                 R.id.ui_percentage_text);
259         ProgressBar bar = (ProgressBar)theRow.findViewById(R.id.ui_bar);
260 
261         // modify the row
262         freqText.setText(sFreq);
263         perText.setText(sPer);
264         durText.setText(sDur);
265         bar.setProgress((int)per);
266 
267         // add it to parent and return
268         parent.addView(theRow);
269         return theRow;
270     }
271 
272     /** Keep updating the state data off the UI thread for slow devices */
273     protected class RefreshStateDataTask extends AsyncTask<Void, Void, Void
    > {
274 
275         /** Stuff to do on a seperate thread */
276         @Override protected Void doInBackground(Void... v) {
```

```
277             CpuStateMonitor monitor = _app.getCpuStateMonitor();
278             try {
279                 monitor.updateStates();
280             } catch (CpuStateMonitorException e) {
281                 Log.e(TAG, "Problem getting CPU states");
282             }
283 
284             return null;
285         }
286 
287         /** Executed on the UI thread right before starting the task */
288         @Override protected void onPreExecute() {
289             log("starting data update");
290             _updatingData = true;
291         }
292 
293         /** Executed on UI thread after task */
294         @Override protected void onPostExecute(Void v) {
295             log("finished data update");
296             _updatingData = false;
297             updateView();
298         }
299     }
300 
301     public boolean getOffsets() {
302         return _updatingData;
303     }
304 
305     public void setOffsets(boolean offsets) {
306         _updatingData = offsets;
307     }
308 
309     /** logging */
310     private void log(String s) {
311         Log.d(TAG, s);
312     }
313 }
```