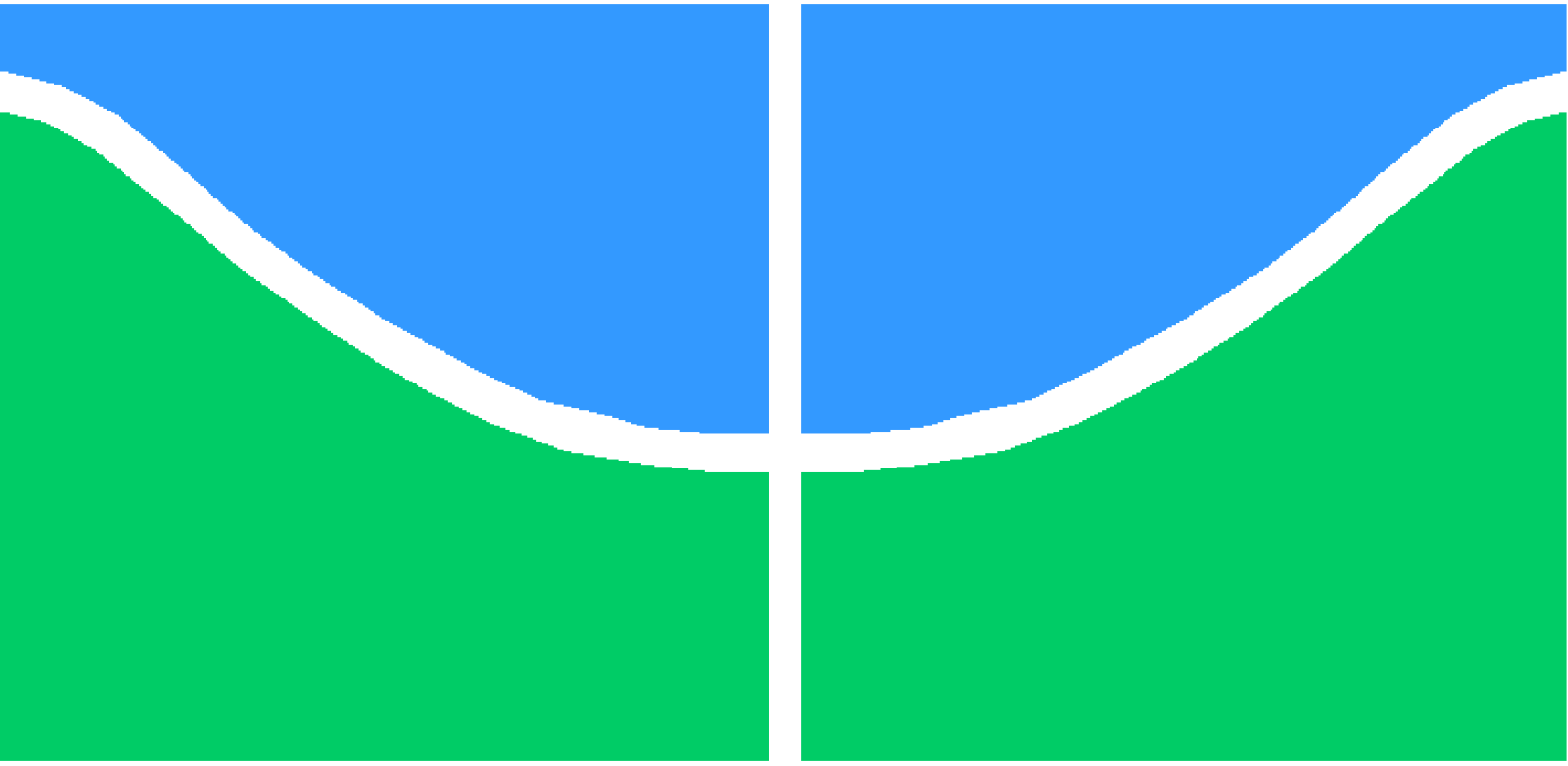

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Fundamentos da engenharia de software aplicados ao desenvolvimento de software livre para Android

Autores: Marcos Ronaldo Pereira Júnior

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2014

Marcos Ronaldo Pereira Júnior

Fundamentos da engenharia de software aplicados ao desenvolvimento de software livre para Android

Monografia submetida ao curso de graduação em Engenharia de Softwareda 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: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2014

Marcos Ronaldo Pereira Júnior
Fundamentos da engenharia de software aplicados ao desenvolvimento de software livre para Android/ Marcos Ronaldo Pereira Júnior. – Brasília, DF, 2014-
44 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Engenharia de Software. 2. Desenvolvimento móvel. 3. Android. I. Prof. Dr. Paulo Roberto Miranda Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Fundamentos da engenharia de software aplicados ao desenvolvimento de software livre para Android

CDU

Marcos Ronaldo Pereira Júnior

Fundamentos da engenharia de software aplicados ao desenvolvimento de software livre para Android

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

Trabalho aprovado. Brasília, DF, Dezembro de 2014:

Prof. Dr. Paulo Roberto Miranda Meirelles
Orientador

Prof. Marcelino Monteiro de Andrade
Convidado 1

Prof. Luiz Augusto Fontes Laranjeira
Convidado 2

Brasília, DF
2014

Resumo

Nos últimos anos, a utilização de dispositivos móveis inteligentes se mostra cada vez mais presente no dia a dia das pessoas, ao ponto de substituir computadores na maioria das funções básicas. Com isso, vem surgindo projetos cada vez mais inovadores que aproveitam da presença constante desses dispositivos, modificando a forma que vemos e executamos várias tarefas cotidianas. Vem crescendo também a utilização de sistemas livres e de código aberto, a exemplo da plataforma Android que atualmente detém extensa fatia do mercado de dispositivos móveis. Ocupando papel fundamental na criação dessas novas aplicações, a engenharia de software traz conceitos de engenharia ao desenvolvimento e manutenção de produtos de software, tornando o processo de desenvolvimento mais organizado e eficiente, sempre com o objetivo de melhorar a qualidade do produto. O objetivo desse trabalho é trazer os fundamentos da engenharia de software para o desenvolvimento de um aplicativo para o sistema Android, aplicando atividades de requisito, desenho, construção, teste e manutenção, bem como o gerenciamento do desenvolvimento como um todo. Como estudo de caso, será utilizado o e-lastic, um dispositivo de controle para exercícios físicos com elástico, tendo como foco a construção de um aplicativo de código aberto para comunicação direta e recepção de dados, gerando *feedback* sobre os exercícios realizados.

Palavras-chaves: Engenharia de Software. Dispositivos móveis. Android.

Lista de ilustrações

Lista de abreviaturas e siglas

UnB	Universidade de Brasília
LED	*Light Emitter Diode*
API	*Application Programming Interface*
JNI	*Java Native Interface*
UID	*User Identification*
DVM	*Dalvik Virtual Machine*
JVM	*Java Virtual Machine*
DEX	*Dalvik Executable*
GUI	*Graphic User Interface*
ICS	*Ice Cream Sandwich*
SWEBoK	*Software Engineering Body of Knowledge*
MVC	*Model-View-Controller*
AIDL	*Android Interface Definition Language*
RPC	*Remote Procedure Call*
AMS	*Activity Manager Service*
SDK	*Software Development Kit*
ADT	*Android Development Tools*
IDE	*Integrated Development Environment*
AVD	*Android Virtual Devices*
App	*Application*
LCD	*Liquid Crystal Display*
MES	Manutenção e Evolução de Software
ESW	Engenharia de Software

Sumário

1 Introdução

1.1 Contexto

A computação eletrônica tem avançado muito desde o século XX, e esse avanço continua bastante visível no século XXI (FONSECA, 2007). Aparelhos e maquinas que antes custavam fortunas e ocupavam salas inteiras puderam avançar ao ponto de caber em uma única caixa. Engrenagens e inúmeras válvulas reduzidas a um micro chip. Várias ferramentas utilizadas no dia a dia de muitos trabalhadores reunidas em um único computador, que adquiriu muitas outras funções além de fazer cálculos.

Nos últimos anos, muitas das funções de um computador vem sendo gradativamente transferidas para dispositivos que cabem na palma da mão (MOORE, 2007). Atualmente podemos encontrar dispositivos móveis com poderosos processadores e alta capacidade de armazenamento de dados, além da conectividade que faz com que a utilização de cabos pareça insensata. A cada dia temos uma maior quantidade de informação concentrada de forma prática e segura, resultando em tarefas cotidianas automatizadas e controladas.

Da mesma forma que cresce a utilização de dispositivos móveis, crescem as empresas os desenvolvem. No mercado atual atuam diversas fabricantes de hardware, mas a grande maioria delas se concentra em três diferentes sistemas operacionais que se destacam no mercado atual: Android, iOS, e WindowsPhone [1]. O Sistema Android controla a maior fatia do mercado de dispositivos móveis, se tornando mais atrativo para desenvolvedores de aplicativos, uma vez que um maior público pode ser alcançado. Por ser o único dos três sistemas que é livre, ele permite uma maior customização e dá mais liberdade para o desenvolvimento que os demais sistemas. Embora já seja um sistema bastante consolidado, existem diversos estudos em todo o mundo sobre como melhorá-lo em diversos aspectos, desde a interface de usuário a complexos módulos de segurança. Esses estudos resultam em contribuições frequentes da comunidade de desenvolvedores, tornando o sistema cada vez mais robusto.

O Android vem avançando em uma velocidade impressionante desde sua criação, e não apenas para celulares. Hoje em dia é possível encontrar o sistema em *tablets*, televisões, relógios, e até mesmo em automóveis.

Inúmeros projetos são desenvolvidos com dispositivos móveis nos dias atuais, em especial no Android. É possível que ao ligar a televisão se veja alguma reportagem sobre algum novo aplicativo que facilita alguma tarefa importante. Diversos projetos inovadores

[1] fonte: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>

surgem em diversas áreas e contextos com a utilização de dispositivos móveis, dentro e fora das universidades. Até mesmo monitores cardíacos podem ser substituídos por um aplicativo com conexão *bluetooth* e um outro dispositivo pequeno para coleta de dados, reduzindo custos.

1.2 Proposta

Equipamentos elásticos vem sendo amplamente utilizados em treinamentos físicos e fisioterápicos (LOSS et al., 2002). Suas vantagens incluem portabilidade, maior liberdade de movimentos e menor custo. Entretanto, esses exercícios não são precisamente controlados fora de laboratórios, e sua utilização é dependente da percepção subjetiva do esforço, com base na sensação de fadiga experimentada durante o exercício, podendo causar danos musculares ao praticante.

O e-lastic é um sistema eletrônico em desenvolvimento que monitora e controla a execução de exercícios físicos realizados com implementos elásticos. Os primeiros protótipos tem um módulo físico de *feedback* para o usuário que apresenta as informações necessárias para o praticante do exercício físico por meio de um display, sinais sonoros e LEDs para sinalização. O protótipo mais recente obtém as informações por sensores da mesma forma que os anteriores, porém sua interface de transmissão dessas informações é a tecnologia *bluetooth*.

Durante este trabalho, será desenvolvido um aplicativo móvel que faz conexão com o dispositivo físico de coleta de informações e controlará o exercício sendo executado. Em adição aos primeiros protótipos, os dados serão apresentados de forma mais clara na tela do usuário e vários tipos de *feedback* para os exercícios podem ser adicionados graças aos recursos encontrados nos dispositivos móveis, como por exemplo a vibração do aparelho. O aplicativo terá um breve tutorial sobre a utilização do aparelho com o implemento elástico, e auxiliará o usuário em todas as etapas do exercício. Os parâmetros serão configurados via *touchscreen* e poderão ser salvos no aplicativo para serem utilizados da próxima vez que o exercício for praticado.

Esse aplicativo será desenvolvido para a plataforma Android utilizando várias técnicas da Engenharia de Software. Todos os processos de coleta e análise de requisitos, design de arquitetura, construção, verificação e validação, e manutenção, serão trabalhados para que se alcance um produto funcional e de qualidade, com a maior eficiência possível. É importante ressaltar que o software desenvolvido será gratuito, totalmente livre e aberto a contribuições de desenvolvedores.

1.3 Objetivos

O objetivo geral deste trabalho é o desenvolvimento e disponibilização do aplicativo em funcionamento na plataforma Android para que seja utilizado em conjunto com o hardware e-lastic, que tem expectativa de comercialização num futuro não muito distante.

objetivos específicos:

- Estudo da plataforma Android e seus componentes;
- Aplicação de conceitos da Engenharia de Software na construção de um aplicativo Android;
- Entrega e distribuição do aplicativo e-lastic na *Play Store*;

O trabalho está dividido em 2 etapas, sendo a primeira etapa tratada neste documento. O objetivo da primeira etapa deste trabalho foi focado no estudo da plataforma e na construção da base arquitetural para o desenvolvimento do aplicativo completo na etapa seguinte, que consistirá no término do desenvolvimento, validação dos resultados, e distribuição do aplicativo.

1.4 Estrutura do trabalho

Este documento está dividido em 3 capítulos. No capítulo 2, é apresentada uma descrição geral da plataforma Android. O capítulo 3 trata da aplicação da engenharia de software no desenvolvimento de um aplicativo para a plataforma Android, apresentando formas de aplicar conceitos utilizando o ambiente de desenvolvimento da plataforma e sua API (*Application programming interface*). O capítulo 4 apresenta mais informações sobre o estudo de caso e alguns resultados e detalhes acerca da primeira etapa de desenvolvimento. Por fim, são apresentadas as considerações finais sobre a utilização da plataforma Android e discussões sobre a primeira etapa do desenvolvimento.

2 Android OS[1]

2.1 Descrição Geral

O Android é um sistema operacional para dispositivos móveis com base em kernel linux modificado, com várias bibliotecas modificadas ou refeitas, de forma a deixar o sistema tão eficiente quanto possível para o hardware limitado que os dispositivos alvo apresentam. A exemplo disso está a biblioteca C *Bionic*, que foi desenvolvida para menor consumo de espaço físico, memória e poder de processamento que as bibliotecas padrão C como a GNU C (glibc) (DEVOS, 2014).

Aplicações desenvolvidas para Android são feitas essencialmente em linguagem Java, com a possibilidade de utilizar outras linguagens como C e C++ através da Java native interface (JNI).

O sistema Android tira vantagem do kernel linux no que diz respeito a identificação e separação de processos rodando no sistema, atribuindo a cada aplicação um *UID* (*User Identification*), e executando cada uma em um processo diferente, isolando umas das outras. Independentemente de a aplicação ser desenvolvida em java ou com código nativo, essa separação de processos do kernel, conhecida como *Application Sandbox*, garante que a aplicação está isolada das demais e portanto sujeita aos mesmos mecanismos de segurança inclusive que os aplicativos do sistema, como contatos, câmera, entre outros.

Cada uma dessas aplicações no sistema funciona em uma instância diferente da Dalvik Virtual Machine (DVM), uma máquina virtual semelhante a Java Virtual Machine (JVM). Códigos em java são compilados e traduzidos para formato .dex (*dalvik executable*), que é executado pela DVM, semelhante ao formato .jar do Java. Essa separação permite que uma falha em um processo de uma aplicação não tenha impacto algum em outra aplicação.

Para interagir com determinados serviços do sistema bem como outras aplicações, uma aplicação deve ter os privilégios correspondentes a ação que deseja executar. Por exemplo, o desenvolvedor pode solicitar ao sistema que sua aplicação tenha acesso a internet, privilégio não concedido por padrão pelo sistema. O usuário então no momento da instalação dessa aplicação é informado que a mesma deseja acesso a internet, e ele deve permitir acesso se quiser concluir a instalação. Todas as permissões requisitadas pelo desenvolvedor e necessárias para o aplicativo realizar suas funções são listadas no momento de instalação, e todas devem ser aceitas, caso contrário a instalação é cancelada. Não é permitido ao usuário selecionar quais permissões ele quer conceder e quais rejeitar

[1] Este capítulo é baseado na documentação oficial disponível em AndroidDevelopers (2014)

à aplicação sendo instalada, tendo apenas as opções de aceitar todas elas, ou rejeitar a instalação.

2.2 Estrutura de uma aplicação

Aplicações no Android são construídas a partir de quatro tipos de componentes principais: *Activities, Services, Broadcast Receivers*, e *Content Providers* (HEUSER et al., 2014).

1. Uma *Activity* é basicamente o código para uma tarefa bem específica a ser realizada pelo usuário, e apresenta uma interface gráfica(*Graphic User Interface*) para a realização dessa tarefa.

2. *Services* são tarefas que são executadas em background, sem interação com o usuário. *Services* podem funcionar no processo principal de uma aplicação ou no seu próprio processo. Um bom exemplo de *services* são os tocadores de músicas. Mesmo que sua interface gráfica não esteja mais visível, é esperado que a música continue a tocar, mesmo se o usuário estiver interagindo com outro aplicativo.

3. *Broadcast Receiver* é um componente que é chamado quando um *Intent* é criado e enviado via broadcast por alguma aplicação ou pelo sistema. *Intents* são mecanismos para comunicação entre processos, podendo informar algum evento, ou transmitir dados de um para o outro. Um aplicativo pode receber um *Intent* criado por outro aplicativo, ou mesmo receber *intents* do próprio sistema, como por exemplo informação de que a bateria está fraca ou de que uma busca por dispositivos *bluetooth* previamente requisitada foi concluída.

4. *Content providers* são componentes que gerenciam o acesso a um conjunto de dados. São utilizados para criar um ponto de acesso a determinada informação para outras aplicações. Para os contatos do sistema, por exemplo, existe um *Content Provider* responsável por gerenciar leitura e escrita desses contatos.

Cada um desses componentes pode funcionar independente dos demais. O sistema Android foi desenvolvido dessa forma para que uma tarefa mais complexa seja concluída com a ajuda e interação de vários desses componentes independente da aplicação a qual eles pertencem, não necessitando que um desenvolvedor crie mecanismos para todas as etapas de uma atividade mais longa do usuário.

Para executar a tarefa de ler um email, por exemplo, um usuário instala um aplicativo de gerenciamento de emails. Então ele deseja abrir um anexo de um email que está em formato PDF. O aplicativo de email não precisa necessariamente prover um leitor de

PDF para que o usuário consiga ter acesso a esse anexo. Ele pode mandar ao sistema a intenção de abrir um arquivo PDF a partir de um *Intent*, e então o sistema encontra um outro componente que pode fazer isso, e o instancia. Caso mais de um seja encontrado, o sistema pergunta para o usuário qual é o componente que ele deseja utilizar. O sistema então invoca uma *Activity* de um outro aplicativo para abrir esse arquivo. Continuando no mesmo exemplo, o usuário clica em um link dentro do arquivo pdf. Esse aplicativo, por sua vez, pode enviar ao sistema a intenção de abrir um endereço web, que mais uma vez encontra um aplicativo capaz de o fazer.

É importante perceber que para uma atividade mais complexa de interação com o usuário, vários aplicativos são envolvidos sem que os mesmos tenham conhecimento dos demais. Cada componente se "registra"no sistema para realizar determinada tarefa, e o sistema se encarrega de encontrar os componentes adequados para cada situação. Esse registro dos componentes é realizado através do AndroidManifest.xml, que é um arquivo incluso em toda aplicação sendo instalada. Ele reúne todos os componentes de uma aplicação, as permissões necessárias para acessar cada um deles, e as permissões que eles utilizam, bem como outras informações.

Uma vez que os componentes de uma aplicação podem ser utilizados por outras aplicações, é necessário um controle maior sobre quem pode ter acesso a cada um deles. Cada desenvolvedor pode criar permissões customizadas para seus componentes, e exigir que o aplicativo que requisite a tarefa tenha essa permissão para acessar o componente. Da mesma forma, o aplicativo que criou essa permissão determina os critérios para conceder a mesma para outros aplicativos. Um simples exemplo de uso desse mecanismo é o fato de uma empresa apenas criar vários aplicativos para tarefas distintas e querer integração entre os mesmos. O desenvolvedor pode definir uma permissão específica para acessar um dos seus *Content Providers*, por exemplo, e definir que apenas aplicativos com a mesma assinatura (assinados pelo mesmo desenvolvedor) possam receber essa permissão. Dessa forma, todos os aplicativos desenvolvidos por essa empresa podem ter acesso aos dados gerenciados por esse *Content Provider*, enquanto as demais aplicações não tem esse acesso.

2.3 Diversidade e Compatibilidade

O Android foi projetado para executar em uma imensa variedade de dispositivos, de telefones a *tablets* e televisões. Isso é muito interessante no ponto de vista do desenvolvedor, que tem como mercado para seu software usuários de diversos dispositivos de diversas marcas diferentes. Entretanto, isso trás uma necessidade de fazer uma interface flexível, que permita que um aplicativo seja utilizável em vários tipos de dispositivos, com vários tamanhos de tela. Para facilitar esse problema, o Android oferece um *framework* em que se pode prover recursos gráficos distintos e específicos para cada configuração de tela,

publicando então um aplicativo apenas que se apresenta de forma diferente dependendo do dispositivo onde ele está sendo executado.

A interface gráfica no Android é essencialmente construída em xml, e tem um padrão de navegação para as aplicações, embora fique a critério do desenvolvedor a aparência de sua aplicação. O desenvolvedor por criar, por exemplo, uma interface gráfica com arquivos xml para cada tamanho de tela, e também diferenciar entre modo paisagem e modo retrato. Entretanto, em se tratando de interface gráfica, vários componentes vão sendo adicionados a API Android ao longo de sua evolução, e portanto vários recursos gráficos necessitam de uma versão mínima do sistema para serem utilizados. Utilizar um recurso presente apenas a partir da versão ICS 4.0.4 (*Ice Cream Sandwich*), por exemplo, implica que o aplicativo não tenha compatibilidade com versões anteriores do sistema.

Da mesma forma, devido a diversa variedade de modelos e fabricantes de hardware, é preciso ficar atento aos recursos de hardware disponíveis para cada dispositivo. Alguns sensores hoje mais comuns aos novos dispositivos sendo lançados no mercado não existiam em modelos mais antigos. O desenvolvedor pode especificar no *Android manifest* os recursos necessários para o funcionamento completo de sua aplicação, de forma que a mesma seja apenas instalada em dispositivos que os apresentarem. Também pode ser feita uma checagem em tempo de execução e apenas desativar uma funcionalidade do aplicativo caso algum recurso de hardware não esteja disponível no dispositivo, se isso for o desejo do desenvolvedor. De forma geral, é relativamente simples a forma com que o desenvolvedor especifica os dispositivos alvo para sua aplicação, tornando essa grande diversidade de dispositivos mais vantajosa do que dispendiosa.

Android é um sistema livre, que pode ser utilizado em modificado e utilizado segundo a licença apache versão 2.2. Shanker e Lal (2011) apresenta alguns tópicos relacionados a portabilidade do sistema Android em um novo hardware. Embora não seja discutida nesse documento, a relativamente fácil portabilidade do sistema para vários tipos de hardware foi uma das razões que levaram seu rápido crescimento, fazendo com que várias fabricantes possam fazer uso do mesmo sistema e lançar vários tipos de dispositivos distintos no mercado, com diferentes *features* e preços, alcançando parcelas do mercado que possuem condições de aquisição muito variadas.

3 ESW e Android

Engenharia de software é definida pela (IEEE, 2014) como aplicação de uma abordagem sistemática, disciplinada e mensurável ao desenvolvimento, operação e manutenção de software. O SWEBoK é um documento bastante consolidado que reúne diversos conceitos relacionados a engenharia de software, e foi um marco para o reconhecimento da engenharia de software como engenharia de fato. Embora sem muitos detalhes, as seções seguintes citam alguns dos processos da engenharia de software que são importantes em todo o ciclo de vida do produto de software, que inclui desde a concepção do produto até a manutenção do mesmo em ambiente de produção.

Segundo Wasserman (2010) alguns aspectos devem ser pensados quando desenvolvendo software para dispositivos móveis:

1. Requisitos de interação com outras aplicações;
2. Manipulação de sensores;
3. Requisitos web que resultam em aplicações hibridas (*mobile* - *web*);
4. Diferentes famílias de hardware;
5. Requisitos de segurança contra aplicações mal intencionadas que comprometem o uso do sistema;
6. Interface de Usuário projetadas para funcionar com diversas formas de interação e seguir padrões de design da plataforma;
7. Teste de aplicações móveis são em geral mais desafiadores pois são realizados de maneira diferente da maneira tradicional;
8. Consumo de energia;

Todos esses tópicos mencionados são muito importantes para o desenvolvimento em várias plataformas móveis, e modificam a forma com que várias atividades da engenharia de software devem ser abordadas.

Holzer e Ondrus (2009) propõe 3 critérios para seleção de plataformas móveis:

1. Remuneração - Relacionado ao número de potenciais clientes que podem adquirir o produto. Plataforma com grande número de usuários e crescimento constante é de grande valia para desenvolvedores. Um ponto centralizado de venda de aplicativos também é um atrativo para a comercialização dos softwares desenvolvidos.

2. Carreira - As oportunidades que o desenvolvimento para a plataforma pode gerar. A possibilidade de trabalhar para grandes e renomadas empresas no mercado pode ser um fator decisivo para a escolha da plataforma. Para aumentar sua credibilidade na comunidade de desenvolvedores e ganhar visibilidade no mercado, Holzer e Ondrus (2009) sugere que o desenvolvedor participe de projetos de software livre, o que é facilitado quando a própria plataforma móvel é aberta.

3. Liberdade - Liberdade criativa para desenvolver. O programador deve sentir que na plataforma ele pode programar o que quiser. Uma plataforma consolidada com excelentes kits de desenvolvimento e dispositivos atrativos do ponto de vista de hardware e sistema operacional atraem muitos desenvolvedores. Plataformas fechadas e com muitas restrições tendem a afastar desenvolvedores que querem essa liberdade, enquanto plataformas abertas apresentam maior liberdade para desenvolvimento.

Com seu grande crescimento e consequente tomada da maior fatia do mercado de dispositivos móveis, a plataforma Android consegue ser bastante atrativa no ponto de vista primeiro tópico, em contraste com as demais plataformas do mercado, como WindowsPhone [1] e iOS [2]. Mais e mais empresas aparecem no contexto do Android tanto em desenvolvimento de hardware quanto software, e as oportunidades de trabalho crescem juntamente com o crescimento da própria plataforma, como sugerido no segundo tópico. Por ser aberto, a plataforma Android também permite que desenvolvedores enviem suas contribuições e correções de bugs ao próprio sistema operacional, aumentando sua visibilidade. Da mesma forma, o Android também apresenta um sistema totalmente aberto e com kits de desenvolvimento consolidados e extensiva documentação disponível online [3], no mínimo se equiparando ao seu principal concorrente iOS em liberdade de desenvolvimento na data de escrita desse documento.

Tomando os critérios apresentados como base, a escolha da plataforma Android para desenvolvimento neste trabalho é feita de forma clara.

3.1 Requisitos

Área de conhecimento em requisitos de software é responsável pela elicitação, análise, especificação e validação de requisitos de software, bem como a manutenção gerenciamento desses requisitos durante todo o ciclo de vida do produto (IEEE, 2014).

Requisitos de software representam as necessidades de um produto, condições que ele deve cumprir para resolver um problema do mundo real que o software pretende atacar. Essas necessidades podem ser funcionalidades que o software deve apresentar

[1] <http://www.windowsphone.com/>
[2] <https://www.apple.com/br/ios/>
[3] <http://developer.android.com/>

para o usuário, chamados requisitos funcionais, ou outras condições que restringem as funcionalidades de alguma forma, seja por exemplo sobre tempo de execução, requisitos de segurança ou outras restrições, conhecidas como requisitos não funcionais.

Requisitos não funcionais são críticos para aplicações móveis, e estas podem precisar se adaptar dinamicamente para prover funcionalidade reduzida (DEHLINGER; DIXON, 2011). Embora o hardware de dispositivos móveis tenha avançado bastante nos últimos anos, dispositivos móveis ainda apresentam capacidade reduzida de processamento devido a limitações como o tamanho reduzido e capacidade limitada de refrigeração. Devido a essas e outras limitações e a grande variedade de dispositivos Android no mercado, com poder computacional bem variado, aplicativos devem ser projetados para funcionar em hardware limitado. Em suma, deve-se pensar sempre em requisitos de performance e baixo consumo de recursos: uso de rede (3g/4g/wifi/bluetooth...), energia, ciclos de processamento, memória, entre outros. Wasserman (2010) afirma que o sucesso de qualquer aplicação, *mobile* ou não, depende de uma grande lista de requisitos não funcionais.

Segundo Dehlinger e Dixon (2011), deve-se analisar bem requisitos de contexto de execução de aplicativos dispositivos móveis. Aplicações móveis apresentam contextos de execução que não eram obtidos em tecnologias anteriores, com dados adicionais como localização, proximidade a outros dispositivos, entre outros, que podem alterar a forma com que os aplicativos são utilizados. Aplicativos móveis tem que ser pensados para se adaptar com essas mudanças de contexto.

O sistema Android permite checar a disponibilidade de recursos de hardware em tempo de execução para que o desenvolvedor possa ajustar as funcionalidades apresentadas ao usuário e prevenir que o usuário encontre problemas na utilização de determinadas funcionalidades. Por exemplo, um jogo simples como *tic tac toe* que utilize *bluetooth* para *multiplayer* pode desativar essa funcionalidade para dispositivos antigos que o não tenham disponível e trabalhar apenas com jogo *single player* contra algum tipo de jogador virtual. Dessa forma é possível prevenir a apresentação para o usuário de funcionalidades que ele na verdade não pode executar.

Requisitos de software podem ser representados de diversas formas, sendo possível a utilização de vários modelos distintos. Na metodologia ágil scrum, por exemplo, os requisitos normalmente são registrados na forma de *User Stories*, onde são geralmente descritos na visão do usuário do sistema. Em outros contextos, podem ser descritos em casos de uso, com descrições textuais e diagramas, ou outras várias formas de representação.

Requisitos de software geralmente tem como fonte o próprio cliente que contrata o serviço do desenvolvimento de software, e são extraídos da descrição de como esse cliente vê o uso do sistema. Todo esse processo é muitas vezes chamado de "engenharia de requisitos".

3.2 Design de software

Desenho de software é o processo de definição da arquitetura, componentes, interfaces, e outras características de um sistema ou um componente (IEEE, 2014). É durante o desenho de software que os requisitos são traduzidos na estrutura que dará base ao software sendo desenvolvido. As necessidades então são traduzidas em modelos, que descrevem os componentes e as interfaces entre os componentes. A partir desses modelos é possível avaliar a validade da solução desenhada e as restrições associadas a mesma, sendo possível avaliar diferentes soluções antes da implementação do software. A partir do design da arquitetura, é possível prever se alguns requisitos elicitados podem ou não ser atingidos com determinada solução, e mudá-la conforme necessário com pouco ou mesmo nenhum custo adicional.

A área de desenho de software pode variar conforme a tecnologia sendo utilizada. A arquitetura do sistema pode variar conforme o sistema operacional alvo, ou mesmo conforme a linguagem de programação que se está utilizando no desenvolvimento. Existem vários princípios de design de software amplamente conhecidos que se aplicam a uma imensidade de situações, sempre com o intuito de encontrar a melhor solução para cada situação e deixar o software modularizado e manutenível.

Aplicativos para o sistema Android são construídos em módulos, utilizando os componentes da API, embora possam ser criadas classes em Java puro sem a utilização de nenhum recurso da API do sistema, e utilizá-las nos componentes assim como em uma aplicação Java padrão desenvolvida para *desktop*. Várias classes de modelo em uma arquitetura MVC (*Model-View-Controller*), por exemplo, possivelmente serão criadas em Java puro.

O Android não impõe nenhuma arquitetura específica no desenvolvimento de aplicações, deixando livre para o desenvolvedor fazer suas escolhas.

Sokolova, Lemercier e Garcia (2013) apresentam alguns tipos de arquitetura derivados do bem difundido MVC, e demonstram uma possibilidade de adaptação do MVC ao Android. Embora a arquitetura MVC possa ser utilizada no Android, ela não é facilmente identificada, e não é intuitiva de ser implementada. *Activities* são os componentes mais difíceis de serem encaixados na arquitetura MVC padrão, embora sejam bem adaptadas às necessidades do desenvolvedor. Por padrão elas tem responsabilidades correspondentes ao *Controller* e ao *View*, e são interpretadas de forma diferente por vários desenvolvedores.

Neste trabalho será utilizada uma arquitetura diferente das derivações MVC apresentadas em Sokolova, Lemercier e Garcia (2013), embora exista o intuito básico de separar a lógica de negócio da apresentação de dados, objetivo principal da arquitetura MVC. Mais detalhes serão apresentados no capítulo 4.

3.2.1 Componentes Android

O Android provê um *framework* para desenvolver aplicativos baseado nos componentes descritos no capítulo 2. Os aplicativos são construídos com qualquer combinação desses componentes, que podem ser utilizados individualmente, sem a presença dos demais. Cada um dos componentes pode ser uma entrada para o aplicativo sendo desenvolvido.

A comunicação direta entre os componentes de cada aplicativo é feita por meio do *Binder*[4], mecanismo de comunicação entre processos. O *Binder* comunica processos através da troca de mensagens (chamadas *parcels*), que podem referenciar dados primitivos e objetos da API assim como referencias para outros objetos *binder*. De forma geral, um *service* no Android pode ter sua interface definida em AIDL (*Android interface definition language*), e uma aplicação que tiver referencia para o *binder* desse *service* pode executar chamadas de procedimento remoto (*RPC - remote procedure calls*) para qualquer método definido nessa interface AIDL. Embora essa forma de comunicação entre processos seja recomendada, o sistema Android também suporta mecanismos de comunicação padrões do Linux como *sockets* e *pipes* (HEUSER et al., 2014). Embora o *binder* apresente acesso direto para alguns componentes, essa comunicação pode ser feita de forma indireta utilizando *Intents*. Como descrito no capítulo 2, Intents são geralmente recebidos por *receivers* que estão registrados para recebê-los. Esse registro é feito por meio de *Intent Filters*. Entretanto, *activities* e *services* também podem utilizar desse mecanismo para ser iniciados e finalizados. Quando um *Intent* é enviado ao sistema via *broadcast*, ele é recebido pelo sistema e resolvido pelo *Activity Manager Service* (AMS), que seleciona o melhor componente para tratá-lo, e então inicia o componente que o recebeu independente da aplicação a que ele pertença. Assim como já descrito, permissões podem ser criadas para restringir essa comunicação, mas a principio qualquer componente pode receber um *intent* de qualquer outra.

Embora os componentes sejam geralmente registrados no sistema, através do *AndroidManifest.xml*, *BroadcastReceivers* podem ser registrados dinamicamente dentro do ciclo de vida da aplicação. Isso quer dizer que é possível criar um *receiver* projetado para funcionar apenas enquanto a aplicação estiver em execução. Esse *BroadcastReceiver* então é registrado por linha de comando da API e desativado quando requisitado, criando uma janela de tempo onde se deseja que esse componente funcione.

De forma geral, para desenhar uma arquitetura para sistema Android deve-se levar em conta todos esses componentes e conhecer bem sua aplicabilidade e a comunicação entre os mesmos. Se for necessário ter algum processo em background independente de *feedback* do usuário, por exemplo, deve-se utilizar do componente *service*. Caso contrário, quando o usuário fechar a interface gráfica do aplicativo, o aplicativo terá sua execução

[4] <http://developer.android.com/guide/components/bound-services.html>

pausada e seu estado salvo para retorno posterior, deixando de executar alguma tarefa que não deveria ter sido interrompida. Como um exemplo mais palpável, o aplicativo do facebook precisa necessariamente utilizar de um serviço para que, mesmo quando não estiver em foco no sistema, notificações de mensagens e atualizações de status cheguem na tela do usuário.

Embora pareça restringir o design de aplicativos, o uso desses componentes unifica a forma com que os aplicativos são desenvolvidos. A API do Android já disponibiliza acesso a vários recursos de hardware do sistema na forma de componentes, e assim como é possível utilizar os componentes do sistema, é possível utilizar componentes de qualquer outra aplicação da mesma maneira - se a permissão for concedida- , e criar os seus próprios componentes para uso de terceiros de forma padronizada. Em suma, é tão fácil usar componentes criados por alguma aplicação qualquer quanto componentes internos do sistema graças ao *framework* de desenvolvimento Android.

3.2.2 Interface de usuário

O design de interface deve ser realizado com o intuito de alcançar usuários com capacidades motoras restritas, e outras limitações como problemas de visão ou audição. Dependendo do público alvo do aplicativo, esses aspectos devem ser considerados.

Com o desafio de fazer uso do pequeno espaço de tela, o design da interface gráfica apresenta mais importância do que jamais teve no desenvolvimento de aplicativos móveis (WASSERMAN, 2010). Usuários estão sempre tentando acessar várias tarefas diferentes, e geralmente tem baixa tolerância a aplicativos instáveis ou não responsivos, mesmo que gratuitos (DEHLINGER; DIXON, 2011).

A base da interface gráfica de usuário no Android é construída em cima da classe *View*, sendo que todos os elementos visíveis na tela, e alguns não visíveis, são derivados dessa classe (BRAHLER, 2010). O sistema Android disponibiliza vários componentes gráficos como botões, caixas de texto, imagens, caixas de seleção de dados, calendário, componentes de mídia (áudio e vídeo), entre outros, e a interface gráfica é construída em cima desses componentes gráficos, que podem ser customizados para um comportamento ou aparência um pouco diferente se assim for desejado, estendendo e customizando suas respectivas classes em Java. Existe um padrão de design[5] que é recomendado que seja seguido.

A interface gráfica do usuário é construída geralmente em formato xml utilizando esses componentes gráficos já disponibilizados na API. Esses arquivos xml são então carregados pela API em Java quando necessário, e podem ser editados dinamicamente através da API, em linguagem Java.

[5] <https://developer.android.com/design/get-started/principles.html>

Todo projeto de aplicativo Android possui um arquivo em Java chamado *R* (*Resources*) autogerado que contém identificação dos recursos do aplicativo. Cada componente gráfico disponibilizado na API e utilizado nos arquivos xml pode ser identificado de qualquer local do aplicativo pelo seu ID atribuído no arquivo xml. Dessa forma, é possível localizar facilmente dentro de sua *Activity* cada componente para ser carregado, modificado e utilizado.

3.3 Construção de software

Essa área de conhecimento é responsável pela codificação do software, por transformar a arquitetura desenhada e seus modelos em código fonte. A construção de software está extremamente ligada ao desenho e às atividades de teste, partindo do primeiro e gerando insumo para o segundo (IEEE, 2014).

Várias medidas podem ser coletadas do próprio código pra auxiliar a avaliação da qualidade do produto sendo construído e gerar insumo para o próprio desenvolvedor reavaliar sua implementação antes da fase de testes.

Para construção de aplicativos para a plataforma Android, o Google disponibiliza um kit de desenvolvimento de aplicativos chamado Android SDK [6] (*Software Development Kit*). Ele provê as bibliotecas da API e as ferramentas necessárias para construir, testar e debugar aplicativos. Android SDK está disponível para os sistemas operacionais Linux, MAC e Windows.

Embora exista uma ferramenta em desenvolvimento (em fase Beta na data de escrita deste documento) para a construção de aplicativos chamada *Android Studio* [7], o Android SDK é distribuído por padrão com a IDE Eclipse, juntamente com *plugin* ADT (*Android Development Tools*), incluindo as ferramentas de visualização de interface em xml, ferramentas de *debug* de código, de análise de arquitetura, níveis de hierarquia de componentes gráficos, testes de desempenho, acesso a memória flash e outros recursos do dispositivo via IDE, entre outras funcionalidades.

Após fazer o download do *Eclipse ADT Bundle*, que contém todas as ferramentas mencionadas, basta fazer o download das APIs desejadas e utilizar do recurso da própria IDE para criar um projeto Android [8], que já vem com uma estrutura pronta para ser trabalhada, separando e categorizando código fonte e outros recursos utilizados no desenvolvimento.

É necessário especificar a versão alvo do sistema para o qual se está desenvolvendo. Para manter a compatibilidade, geralmente desenvolvedores utilizam como versão

[6] <https://developer.android.com/sdk>
[7] <https://developer.android.com/sdk/installing/studio.html>
[8] <https://developer.android.com/tools/projects/index.html>

mínima a 2.3 ou anterior, embora para utilizar alguns recursos seja necessário utilizar uma API mínima mais recente. Todas as APIs utilizadas precisam ser baixadas pela própria ferramenta de download do SDK. Para desenvolver para Android kitkat 4.4.2 com compatibilidade com Android 2.3 por exemplo, deve-se obter ambas APIs.

Juntamente com o SDK, o desenvolvedor tem acesso a uma ferramenta de criação de dispositivos virtuais AVD (*Android Virtual Devices*) para poder executar as aplicações sendo desenvolvidas sem a necessidade de um dispositivo físico disponível para esse fim.

3.4 Testes

Testes de software consistem em verificar se o produto de software se comporta da forma esperada em um determinado conjunto de casos específicos, selecionados com o intuito de representar o maior número de situações diferentes que podem ocorrer durante o uso do sistema, com o software em execução. Os testes têm que ser projetados para checar se o software está de acordo com as necessidades do usuário, procedimento conhecido como validação, e para verificar se as funcionalidades estão de acordo com a especificação, procedimento conhecido como verificação (IEEE, 2014). Testes podem ser realizados em vários níveis, desde o teste de pequenos trechos de código até a interação entre componentes e o teste da interface gráfica do usuário.

Existem vários tipos de testes aplicáveis a determinados tipos de sistema. De acordo com as necessidades e o ambiente onde o sistema irá funcionar, vários testes podem ser ou não necessários para garantir o funcionamento do sistema sob diversas condições. Sistemas web podem exigir testes de carga e stress para avaliar a quantidade de usuários simultâneos suportados, por exemplo. Sistemas críticos já também necessitam de testes de recuperação, para avaliar a capacidade do sistema de manter ou restaurar seu funcionamento após algum tipo de falta.

Ter uma boa suíte de testes é de grande valia para a manutenção de um produto de software, uma vez que sempre que uma modificação precisar ser feita no sistema, é possível verificar de forma automatizada se algum comportamento foi indevidamente alterado pela modificação realizada.

O sistema Android provê um *framework* para testes[9] , com várias ferramentas que ajudam a testar o aplicativo sendo desenvolvido em vários níveis, desde testes unitários a testes relacionados ao *framework* de desenvolvimento e a testes de interface de usuário. Todas as ferramentas necessárias para utilizar a API de testes disponível no *framework* de desenvolvimento Android são disponibilizadas juntamente com o Android SDK.

Podem existir classes em Java puro que não utilizam a API de desenvolvimento

[9] <http://developer.android.com/tools/testing/testing_android.html>

do Android. Conseguir fazer essa separação de classes que utilizam o *framework* e classes em Java puro pode significar uma maior facilidade em testar determinadas partes da aplicação, uma vez que podem ser diretamente utilizados os já bem difundidos *JUnit tests* para essas classes. Entretanto, muitas classes são construídas através dos componentes, e o funcionamento das mesmas também precisa ser testado.

As suítes de teste no Android são baseadas em *JUnit*, e então da mesma forma que é possível utilizar *JUnit* para desenvolver testes para classes que não utilizam a API Android, é possível utilizar as extensões do *JUnit* criadas especificamente para testar cada componente do aplicativo sendo desenvolvido. Existem extensões *JUnit* específicas para cada componente Android, e essas classes contém métodos auxiliares para criar *Mock Objects*. Estes são criados para simular outros objetos do contexto de execução real do aplicativo.

O kit de desenvolvimento para a plataforma Android inclui ferramentas automatizadas de teste de interface gráfica. O robotium, por exemplo, realiza testes de caixa preta em cima da interface gráfica do aplicativo. São criadas rotinas de teste em Java semelhantes ao *JUnit*, com *asserts* para validar os resultados. Com ele é possível criar rotinas robustas de testes para validar critérios de aceitação pré-definidos, simulando interações do usuário com uma ou várias *activities*. Existe a ferramenta Monkeyrunner, onde se cria *scripts* em Python para instalar e executar algum aplicativo, enviando comandos e cliques para o mesmo, e salvando *screenshots* do dispositivo no computador com resultados. Há também a ferramenta Monkey, que é utilizada para fazer testes de stress no aplicativo gerando inputs pseudo aleatórios.

3.5 Manutenção

A manutenção de software trata dos esforços de desenvolvimento com o software já em produção, isto é, em funcionamento no seu devido ambiente. Problemas que passaram despercebidos durante as fases de construção e testes são encontrados e corrigidos durante a manutenção do sistema. Da mesma forma, o usuário pode requisitar novas funcionalidades que ele não havia pensado antes do uso do sistema, e o desenvolvimento dessas novas funcionalidades é também tratado como manutenção uma vez que o software já se encontra em produção, num processo conhecido como evolução de software. Revisões de código e esforços com manutenibilidade também podem ser consideradas atividades de manutenção, embora possam acontecer antes do sistema entrar em produção.

A manutenção de software geralmente acontece por período mais longo que as demais fases do desenvolvimento do software citadas nos tópicos anteriores, ocupando a maior parte do ciclo de vida do produto.

A separação de componentes independentes apresentados neste capítulo é de grande

valia para a manutenibilidade do sistema. Essa separação permite que componentes tenham sua funcionalidade específica melhor compreendida e possam ser substituídos sem grandes impactos nos demais.

Atividades de design de software devem ser reforçadas para garantir uma fase de manutenção com a menor quantidade de problemas possível. Uma arquitetura modularizada é mais fácil de ser entendida e modificada e consequentemente mantida. Também é importante que se utilize de padrões de codificação, identação e documentação para que a manutenção seja facilitada inclusive para desenvolvedores que não tem familiaridade com o código desenvolvido. Empresas tendem a colocar equivocadamente engenheiros junior para dar manutenção a sistemas e engenheiros mais experientes para desenvolver novos projetos, e adotar práticas de desenvolvimento que agem em favor à manutenibilidade ajudam a amenizar os problemas causados por esse tipo de alocação de recursos humanos.

Sendo um sistema de código aberto, o Android permite que o desenvolvedor possa o analisar e consequentemente o entender a ponto de corrigir *bugs* do sistema, criar funcionalidades novas, e também portá-lo para novos *hardwares* (GANDHEWAR; SHEIKH, 2010). Isso é um ponto importante para atividades de manutenção do sistema Android como um todo, e permitiu que a plataforma crescesse de forma extremamente acelerada em um pequeno espaço de tempo. Um dos motivos para a própria plataforma Linux estar em um estado tão estável nos dias atuais foi os anos que a mesma teve de contribuição da comunidade sendo um software livre e portanto tendo seu código aberto a qualquer desenvolvedor, assim como o sistema Android.

4 Estudo de Caso

4.1 E-Lastic

O E-lastic é um sistema eletrônico que monitora e controla a execução de exercícios físicos realizados com equipamento que impõe sobrecarga à movimentação de segmentos corporais por meio de resistência elástica. Geralmente o controle de sobrecarga gerada por elementos elásticos é baseado na percepção subjetiva do esforço, com base na sensação de fadiga experimentada durante o exercício, e portanto o praticante não têm controle do esforço aplicado no exercício.

O produto em desenvolvimento apresenta um aparelho portátil, voltado para o controle de atividades físicas em ambientes fechados ou abertos. Trata-se de um sistema eletrônico embarcado que realiza o processamento digital do sinal originado num sensor de força e associa essas informações com variáveis de espaço e tempo, de forma a gerar informações suficientes para o controle e prescrição de exercícios resistidos. O sensor de força será fornecido calibrado juntamente com a central de processamento. Esse sistema eletrônico permite a acoplagem do implemento elástico para a realização do exercício a ser monitorado, e interfaceia com o usuário por meio de um aplicativo desenvolvido para um dispositivo móvel. De forma simplificada, durante o exercício físico, a força aplicada pelo usuário ao elemento elástico é calculada no microcontrolador e enviada juntamente com as demais informações via *Bluetooth* para um dispositivo móvel com o e-lastic *app*, que contém opções de controle para a realização do exercício físico.

Para a utilização do aparelho, o usuário deve selecionar o implemento elástico mais adequado ao exercício que será praticado, com resistência elástica compatível com o esforço que será aplicado. Um sensor de força compatível é acoplado ao implemento elástico e ligado ao módulo de processamento por uma entrada específica. O usuário então seleciona via aplicativo móvel um dos três modos de execução de exercício: dinâmico, isométrico e potência. Seleciona em seguida a configuração da sua rotina de treino, com os parâmetros de controle para o exercício específico. A partir daí o exercício pode ser iniciado, e informações precisas de controle são apresentadas na tela do dispositivo móvel em tempo real. Vibração do dispositivo, estímulos sonoros e outros tipos de sinais podem ser utilizados para dar *biofeedback* ao usuário sobre o exercício em curso. Após a realização do exercício, é apresentado um resumo da seção de treino executada. É possível que se apresente um histórico de treinamento dos indivíduos que realizam suas atividades utilizando o e-lastic.

Os parâmetros para controle podem variar de exercício para exercício, porém em

todos eles será selecionado um intervalo de intensidade, isto é, carga máxima e mínima em quilogramas, em que o exercício será trabalhado. As opções de exercícios são espécies de rotinas pré programadas, modos de utilização da resistência elástica para o exercício físico:

1. Modo dinâmico - Realização de movimentos cíclicos em velocidade lenta e cadência controlada. Os parâmetros de controle para esse exercício são número de séries de exercícios, número de repetições por série e a duração do descanso entre as séries.

2. Modo isométrico - Contrações estáticas em que o usuário mantém uma posição e sustenta a sobrecarga elástica nessa posição por um determinado tempo. Semelhante ao modo dinâmico, no modo isométrico o usuário seleciona os parâmetros de números de contrações realizadas (repetições do exercício), e duração do tempo de intervalo entre as séries, com adição da duração de cada contração, isto é, o tempo em que o esforço deve ser mantido para que uma repetição seja contabilizada.

3. Modo potência - Semelhante ao modo dinâmico, estes são exercícios cíclicos, porém com a maior velocidade possível. Os parâmetros para esse exercício são os mesmos do modo dinâmico, porém aqui a velocidade do movimento faz parte dos parâmetros de controle.

4.2 Desenvolvimento

O e-lastic começou como um produto desenvolvido para interfacear com usuário por modo de um dispositivo específico conectado ao sensor de força via cabo. A seleção dos exercícios e o próprio *feedback* para o usuário eram realizados por meio desse dispositivo, fazendo interação com o usuário por meio de um display LCD e um potenciômetro para seleção de parâmetros de exercício, e o *feedback* sendo feito por meio de LEDs e sinais sonoros, bem como o próprio display LCD. Esse produto é resultado do trabalho de mestrado em processamento de sinais da aluna Fernanda Sampaio Teles, que resultou num pedido de patente com número de registro BR 5120130007631.

Por volta de 1 vez ao mês, foram realizadas reuniões com Fernanda e outros integrantes da equipe. Daqui em diante serão referenciados neste documento como *product owners*.

Os requisitos para o aplicativo foram a principio retirados do próprio comportamento dos primeiros protótipos. Esses comportamentos foram descritos e detalhados pelos *product owners*, que não só descreveram como os protótipos funcionavam, como também foi discutido como o aplicativo deveria se comportar de forma diferente dos mesmos, com adições de novas funcionalidades para o aplicativo sendo desenvolvido.

A ideia inicial do aplicativo, e foco principal do desenvolvimento da equipe ao longo desse semestre, foi fazer com que o aplicativo sendo desenvolvido para plataforma Android tenha no mínimo as mesmas funcionalidades dos primeiros protótipos.

O desenvolvimento do e-lastic app ao longo do semestre foi planejado e executado juntamente com a disciplina de Manutenção e Evolução de Software (MES). Desde o início do semestre, foi preparada uma equipe na disciplina para o desenvolvimento do aplicativo desde o início, de forma a preparar a base da arquitetura. Essa base será utilizada na segunda fase deste trabalho, quando o desenvolvimento será focado em alcançar todos os objetivos traçados nas reuniões com os *product owners*, que se resumem basicamente na construção e entrega do aplicativo completo e funcional.

Para este trabalho em conjunto com a disciplina de MES durante o semestre, o objetivo principal é conseguir uma base de arquitetura e alcançar dois tipos de exercícios com funcionalidade implementada: exercício dinâmico e isométrico. A interface gráfica foi deixada em segundo plano durante esse período de desenvolvimento, com o foco voltado então em uma base estável e manutenível com as funcionalidades básicas implementadas e com seus respectivos testes.

4.2.1 Ciclo de desenvolvimento

Durante o desenvolvimento, foram utilizadas práticas ágeis retiradas do Scrum. Embora o Scrum não esteja no escopo deste trabalho, informações sobre o mesmo podem ser obtidas em Schwaber e Beedle (2002).

Devido ao ritmo de trabalho mais lento proporcionado pelo tempo curto da disciplina de MES, as *sprints* foram planejadas para durar 15 dias (2 semanas). Inicialmente foram criadas várias histórias de usuário (*user stories*) e montado um *backlog* considerando a reunião inicial com os *product owners*. Novas histórias surgiam, bem como histórias iam sendo atualizadas conforme os requisitos ficavam mais claros com as reuniões subsequentes. A cada 2 semanas então eram consideradas as histórias de usuário do *backlog* e priorizadas para a iniciar a *sprint*. Nas medidas utilizadas pela equipe, com pontuação de 1 a 5 para cada história utilizando técnica do *planning poker*, foi implementada uma média de 7 *story points* por *sprint*. É importante ressaltar que histórias não completadas não entram nem parcialmente nesse cálculo e são jogadas para as *sprints* seguintes, e portanto para uma equipe inexperiente em constante aprendizado é possível inferir que mais esforço foi dedicado do que o próprio *velocity* pode indicar, representando a curva de aprendizado da equipe. Muito do esforço do principal autor deste trabalho foi direcionado ao aprendizado da equipe para alavancar o desenvolvimento do aplicativo.

Todo trabalho realizado foi feito utilizando técnica de programação em pares. Um quadro mapeando os integrantes que já trabalharam juntos esteve em constante atua-

lização para que a equipe circulasse da melhor forma possível. O principal autor deste trabalho participou do desenvolvimento dessa forma várias vezes ao longo do semestre, mesmo esse não sendo o objetivo principal, com o intuito de compartilhar de parte do conhecimento da plataforma obtido em projeto com parceria UnB-Positivo de desenvolvimento de aplicativos Android, bem como conhecimento adquirido em estágio no ICRI-SC (*Intel Collaborative Research Institute for Security Computing*), focado em segurança móvel.

O código sendo desenvolvido se encontra disponível em um repositório aberto[1] para qualquer desenvolvedor através da plataforma gitlab.com, e a equipe de alunos de MES não tem acesso direto de escrita para esse repositório. Para contribuir para o projeto, a equipe trabalha em uma cópia do repositório original (*fork*). Quando alcança algo significativo para o projeto, a equipe então cria um *merge request* pela própria plataforma gitlab, que é então avaliado e aceito pelos revisores, a exemplo do principal autor deste trabalho, ou rejeitado e mudanças são solicitadas. O mesmo procedimento pode ser realizado por qualquer desenvolvedor que deseja contribuir com o projeto. Como já dito, o software sendo desenvolvido é livre e portanto tem código aberto disponibilizado online. A única parte do produto e-lastic que será comercializada é o componente físico, com o módulo central de processamento, sensor de força e implemento elástico.

4.2.2 Equipe

Composta por 5 alunos da disciplina de MES, a equipe contém apenas graduandos em Engenharia de Software da Universidade de Brasília campus UnB Gama. Todos eles tem um tempo a dedicar ao desenvolvimento correspondente ao tempo de uma disciplina de 4 créditos, com 4 horas-aula e mais 4 horas semanais adicionais por integrante, totalizando 8 horas por semana de dedicação para cada um dos integrantes da equipe.

Todos os integrantes não tinham a priori nenhuma experiência no desenvolvimento de aplicativos para a plataforma Android, porém todos eles tinham algum conhecimento em linguagem de programação Java, necessária ao desenvolvimento, bem como experiência em atividades de verificação e validação, banco de dados, entre outras áreas. Todo esse conhecimento foi adquirido ao longo do curso de graduação em Engenharia de Software que está em andamento, provendo o background necessário para que eles pudessem trabalhar neste projeto durante o semestre. A equipe também apresentou conhecimento suficiente em gerência de configuração e controle de versionamento, o que ajudou significativamente em relação a manutenção do repositório do projeto.

Um dos integrantes da equipe faz o papel de *coach*, coordenando o restante da equipe para alcançar os objetivos de cada *sprint* e tomando frente da organização da equipe ao longo dos ciclos de desenvolvimento. Embora tenha essa responsabilidade adi-

[1] <http://gitlab.com/biodyn/biodynapp>

cional, esse integrante participava das atividades de desenvolvimento tanto quanto os demais, e foi selecionado para esse papel pelo professor da disciplina, orientador deste trabalho, por ter mais experiência que os demais em metodologia ágil de desenvolvimento de software.

O principal papel exercido pelo principal autor deste trabalho ao longo do desenvolvimento foi em relação ao estabelecimento da arquitetura do aplicativo sendo desenvolvido, visto que a equipe não tinha experiência com componentes do Android e não conhecia a própria estrutura de um aplicativo para essa plataforma. Dessa forma, ele ficou responsável por criar a estrutura base para o desenvolvimento do aplicativo, bem como revisar todos os códigos submetidos para o repositório principal, e solicitar modificações caso não estivessem de acordo com a arquitetura base do repositório principal, ou não estivesse com qualidade aceitável. Como o autor participou ativamente do desenvolvimento, não houve muita surpresa nos *merge requests* e portanto poucos foram rejeitados.

Durante as aulas da disciplina de MES, toda a equipe esteve presente com o principal autor deste documento, onde era aproveitado o tempo em conjunto para a disseminação do conhecimento sobre a plataforma. Por várias vezes foram feitos DOJOs de treinamento para que o conhecimento circulasse dentro da equipe. Com o mesmo objetivo, a equipe sempre trabalhou em revezamento de pares e de tarefas. Uma dupla que trabalhou com um módulo específico durante uma semana passava a trabalhar em outro na semana seguinte, ou mesmo em atividades de teste, com outra dupla assumindo o trabalho da anterior. Os demais horários da equipe variam de acordo com a semana e com a dupla que trabalhará na semana. As duplas sempre alternavam, e os horários para cada dupla dependem da disponibilidade dos integrantes. Nesses horários de desenvolvimento, o autor deste trabalho não participou de forma presencial e não houve participação de toda a equipe reunida, cada dupla trabalhava em uma parte separada da aplicação.

Como os conhecimentos em desenvolvimento Android foram adquiridos ao longo do desenvolvimento, a primeira *sprint* teve menor *velocity* que as demais.

Embora seu principal papel fosse como arquiteto e revisor de código, o autor deste trabalho também participou da codificação do aplicativo, codificando e construindo boa parte dos principais componentes desenvolvidos ao longo do semestre. Um dos motivos para isso foi dar o impulso inicial que a equipe inexperiente na plataforma precisava para trabalhar de forma mais eficiente.

4.3 Estado da Arquitetura

O planejamento do desenvolvimento para a equipe da disciplina de MES foi focado na base da arquitetura, juntamente com a funcionalidade básica do aplicativo. Isso inclui o modo de exercício dinâmico e exercício isométrico com seus respectivos testes funcio-

nais, e a comunicação com o microcontrolador via *Bluetooth* para o funcionamento dos exercícios. A interface gráfica de usuário foi deixada em segundo plano durante essa etapa do desenvolvimento, tendo apenas alguns componentes temporários apresentados na tela para ajudar na visualização da funcionalidade que foi desenvolvida. Ainda durante esse período de colaboração com a equipe de MES, também deve ser entregue a persistência dos parâmetros de execução dos exercícios, de forma que o usuário possa salvar suas preferências e carregá-las para reutilizar os mesmo parâmetros da próxima vez que praticar o exercício.

Nesta seção, serão chamados de componentes instanciações dos componentes Android, e de módulos conjunto mais complexo de classes que envolvem um ou mais componentes. Na implementação atual do aplicativo, exitem os seguintes componentes e módulos principais:

- *BluetoothService* - Responsável pela conexão *bluetooth*, este componente é um *Android Service* que se inicia juntamente com a aplicação e fica a espera de uma solicitação de conexão vinda da tela de escolha de dispositivos. Quando o usuário solicita a conexão com o hardware e-lastic selecionando-o em uma lista, o aplicativo identifica o dispositivo a ser conectado e informa ao *BluetoothService*, que a partir daí é responsável por conectar e manter a conexão, recebendo os dados do microcontrolador presente no hardware e-lastic.

- *ExerciseService* - Este módulo reúne a maior parte da lógica de negócio. Ele gerencia qual é o exercício ativo e trata da execução do mesmo. Quando recebe um dado de força vindo do *bluetoothService*, o *exerciseService* informa ao exercício ativo os novos valores recebidos. Da mesma forma ele informa ao exercício sobre mudanças nos parâmetros vindas da interação com o usuário com o a tela do aplicativo.

- *BioFeedbackService* - Este módulo é responsável por gerar *biofeedback* para o usuário. Quando o valor de força é superior ao limite máximo, por exemplo, o exercício informa a esse componente que algum tipo de *feedback* deve ser acionado para informar ao usuário que o limite foi ultrapassado. Por sua vez, este componente checa sua lista de *biofeedbacks* e aciona os que foram solicitados. Por exemplo, pode ser solicitado que o dispositivo móvel vibre quando os limites de força forem violados, como também pode ser requisitado que um som seja tocado. Essas configurações de que tipo de *biofeedback* ativar estarão disponíveis para o usuário na versão final do aplicativo.

- *ExerciseActivity* - Representa a tela principal e a própria interação do usuário com os componentes gráficos. Este componente não será totalmente implementado neste período de desenvolvimento com parceria com a equipe de Manutenção e Evolução de software, e será trabalhado apenas na segunda etapa deste trabalho. Neste período,

apenas alguns componentes gráficos temporários estão disponíveis para demonstrar a funcionalidade básica do aplicativo em execução.

A comunicação de todos os componentes e módulos é feita de forma indireta por meio de *intents*. Como já citado em capítulos anteriores, o *intent* tem a função de comunicar componentes independente da aplicação a qual pertencem. Entretanto, neste aplicativo em desenvolvimento, essa comunicação deve ser feita apenas entre os componentes internos da aplicação, não sendo necessário que esses *intents* entre os componentes internos sejam enviados pelo sistema Android para outros componentes de outras aplicações. Para esse fim, a implementação foi feita utilizado um recurso da API Android chamado *LocalBroadcastManager*, que é responsável por realizar o envio de *broadcast intents* apenas para os componentes internos da aplicação, evitando que os mesmos sejam recebidos em outras aplicações. De forma análoga, os receptores desses *intents* são registrados não diretamente no sistema, mas dentro dessa instância de *LocalBroadcastManager* que é individual da aplicação, registrando-se apenas para receber *intents* enviados dentro da própria aplicação. O uso desse recurso evita a necessidade de criar permissões específicas para cada tipo de *intent* que será utilizado entro da aplicação, e não há perigo de que a aplicação receba *intents* forjados por alguma outra aplicação maliciosa ou mesmo que alguma aplicação maliciosa receba dados que são privados deste aplicativo.

Toda essa comunicação indireta foi reunida em uma classe responsável por gerenciar essas comunicações, a classe *Communicator*. Essa classe reúne todas as *actions* de todos os *intents* que as classes estão preparadas para receber. Basicamente, para saber que informações o *ExerciseService* espera, por exemplo, basta checar as constantes na classe *Communicator.ExerciseServiceActions*. Embora essa classe pareça ter um alto acoplamento devido a sua responsabilidade de comunicar todos os componentes, esse acoplamento é reduzido devido a comunicação indireta que é feita com o uso de *intents*. Simplificando, a classe *Communicator* tem a responsabilidade de enviar mensagens para todos os componentes sem realmente os conhecer. Se um componente for alterado, não há impacto algum dentro dessa classe.

Essa comunicação indireta que é implementada por meio dos *intents* tem a vantagem de deixar os componentes com uma conexão bastante fraca. Quando um componente envia uma mensagem, ele não sabe ao certo quem vai receber, portanto não precisa conhecer o receptor. As mensagens são enviadas via *broadcast*, para todos os componentes, e cada um recebe o que desejar receber. Da mesma forma, os receptores dessas mensagens não sabem quem as mandou, e portanto não há uma associação direta em nenhum dos lados da comunicação. Embora pareça deixar a comunicação confusa e embaralhada, isso permite que troquemos um componente inteiro do projeto com pequeno ou nenhum impacto nos demais componentes. A interface gráfica, por exemplo, recebe mensagens do componente de exercício informando sobre atualizações nos dados. Ela não conhece o

exercício em execução e o exercício em execução não conhece quem está mostrando seus dados. Se a interface gráfica fosse inteiramente excluída da aplicação, com pouquíssimos ajustes, relacionados principalmente as mudanças no próprio *AndroidManifest.xml*, seria possível deixar várias outras funcionalidades ainda em funcionamento. Com esse acoplamento reduzido entre os componentes da aplicação, se torna muito mais fácil a manutenção e refatoração desses componentes isoladamente, ou mesmo a substituição completa dos mesmos. Trocar o módulo *bluetooth* por um módulo *wifi*, por exemplo, que recebe esses dados por *socket* de rede em vez de *bluetooth*, não traria quase nenhum impacto a aplicação já em produção, uma vez que esse novo módulo poderia simplesmente enviar mensagens da mesma forma que o *bluetooth* enviava utilizando a classe *Communicator*. Como já explicitado, para os demais módulos não importa quem envie os dados, desde que eles cheguem corretamente.

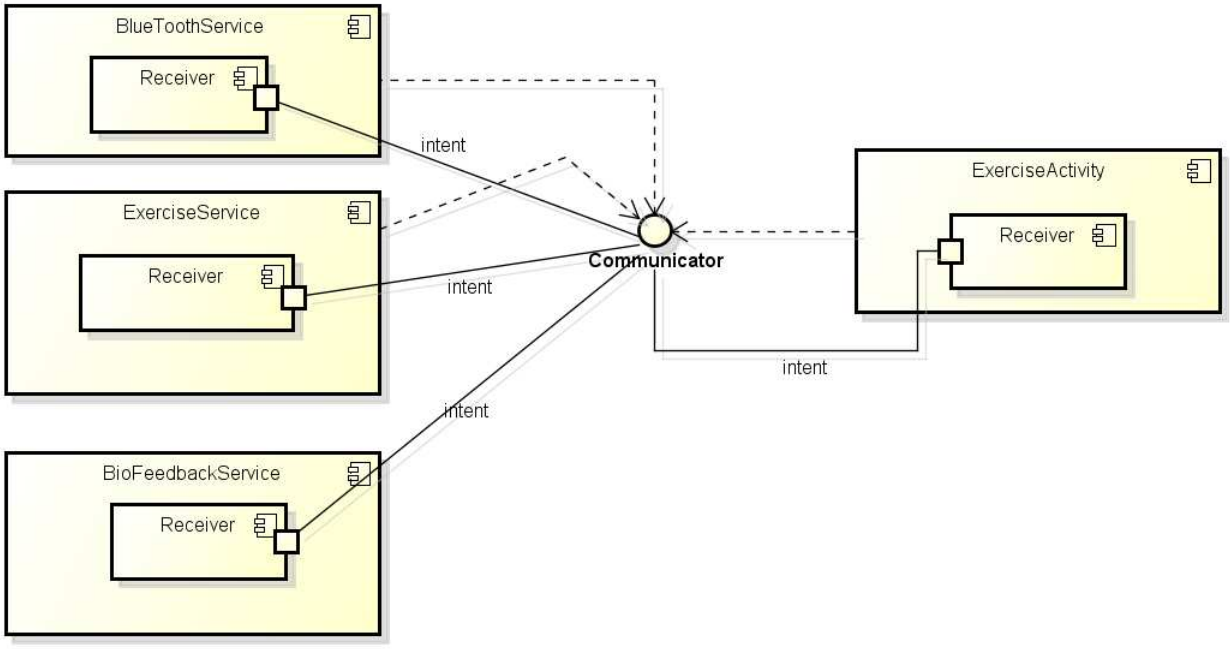

Figura 1 – Componentes e sua comunicação via classe *Communicator*

A única exceção para essa comunicação indireta é a própria inicialização dos componentes. Todos os serviços precisam ser iniciados em algum lugar na aplicação, e o lugar óbvio para o fazer é no início da aplicação. Não tem porque iniciar os componentes enquanto o usuário não abrir a parte gráfica da aplicação, e por esse motivo os serviços são inicializados assim que a *Activity* principal é criada, já que ela é o ponto de entrada da aplicação e-lastic. Muitos componentes podem ser o ponto de entrada em um aplicativo Android, mas neste caso específico a melhor escolha é a própria interface gráfica, representada pela *Activity*. Desse modo a *Activity* conhece os serviços a serem inicializados, porém não conhece seu comportamento e nem mesmo sua responsabilidade.

A Figura 1 é um diagrama de componentes que representa de forma gráfica a ideia

básica da comunicação entre os componentes da aplicação e-lastic.

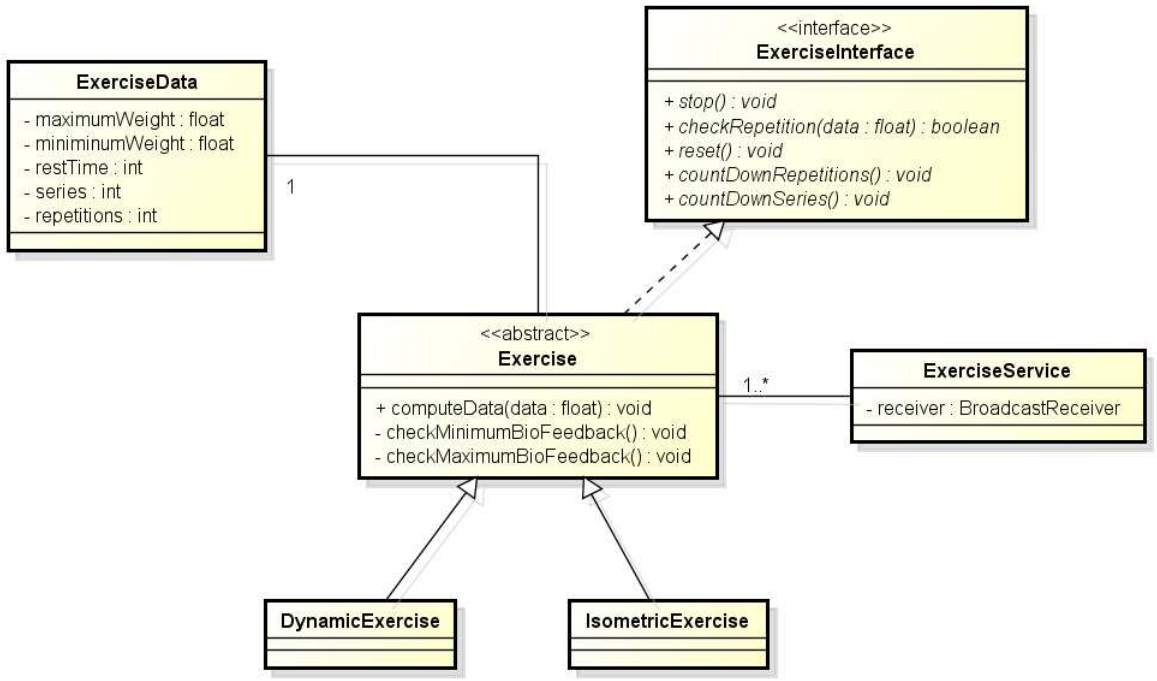

Figura 2 – Principais classes dentro do módulo de exercício e suas relações

Embora representado no diagrama de componentes como uma caixa apenas (*ExerciseService*), o módulo de exercício contém a lógica da comunicação do componente via *Communicator* e o gerenciamento do exercício em execução. É utilizada a generalização para que o *service* tenha o mesmo comportamento independente do exercício em execução, e portanto as manipulações do exercício são feitas em cima de um objeto do tipo *Exercise*, que é uma classe abstrata. A Figura 2 contém um diagrama de classes para demonstrar a estrutura básica desse módulo.

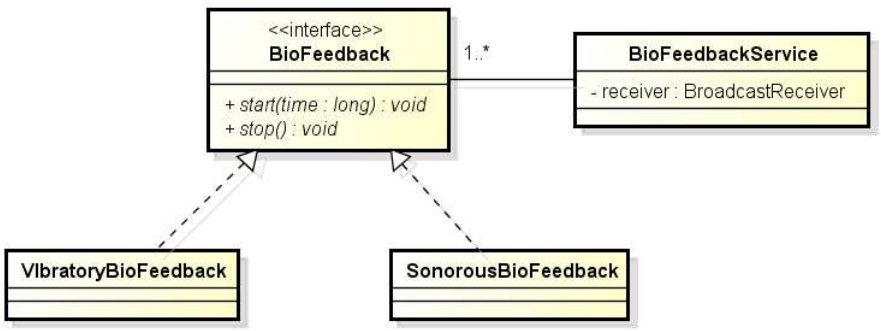

Figura 3 – Principais classes dentro do módulo de *biofeedback* e suas relações

As classes internas do módulo de *biofeedback* podem ser vistas na figura 3. Também é utilizada uma generalização dos tipos de *biofeedback* com a interface *BioFeedback*, que deve ser implementada para criar um novo tipo de *biofeedback*. Da mesma forma que o módulo de exercício, aqui também existe um *service* que fica em segundo plano a espera de uma requisição de execução de *biofeedback*. Em suma, ele fica em *standby* até que algum pedaço da aplicação solicite que ele ative algum dos seus tipos de *feedback*. Reunir essa função em um módulo específico da aplicação faz com que se possa ter controle de quais tipos de *biofeedback* podem ser utilizados, e eles tem um acesso padrão independente do tipo, com a utilização do *Communicator* e *flags* que identificam que tipos de *biofeedback* ativar. Tocar um som e vibrar o celular são feitos de formas completamente diferentes, e só a própria classe precisa saber como o fazer, sendo que o *service* só conhece a interface de início e término e para ela não importa sua implementação. Com a implementação dessa forma, basta uma adição de uma linha de código no mapa de *biofeedbacks* e a adição de uma *flag*, além da implementação do novo *biofeedback*, para que um novo tipo de *biofeedback* esteja pronto para ser utilizado.

As explicações neste tópico tem como objetivo principal tentar demonstrar o tipo de comunicação entre as partes da aplicação e a modularização da mesma, justificando as principais decisões acerca da arquitetura e demonstrando a preocupação da equipe com manutenibilidade e princípios de design. Não serão explorados todos os detalhes da implementação da aplicação desenvolvida, sendo que isso será o alvo da segunda parte deste trabalho.

5 Considerações finais

Grande parte da literatura ainda referencia diretamente o guia de desenvolvimento da plataforma Android, e não há contribuições significativas que já não estejam contempladas no próprio guia. Por esse motivo, várias fontes de leitura apenas apresentam informações redundantes e portanto não foram referenciadas neste documento. As principais fontes aqui referenciadas contemplam trabalhos de pesquisa em segurança, estudo e análise de arquitetura, ou outras temáticas relacionadas não apenas ao Android, mas à dispositivos móveis em geral e à engenharia de software.

Como plataforma de desenvolvimento, o Android apresenta um ambiente bastante facilitado, com uma curva de aprendizado relativamente rápida para desenvolvedores familiarizados com linguagem de programação Java. Sua extensa documentação online, com o guia de desenvolvimento bastante detalhado e várias comunidades de desenvolvedores, como o *stackOverflow*, que vem crescendo em uma rápida proporção juntamente com o avanço da própria plataforma, apresentam aos novos desenvolvedores uma forma rápida de aprender com os desenvolvedores mais experientes e também com os próprios criadores da plataforma, que também participam de fóruns de desenvolvedores e ajudam a esclarecer diversos pontos sobre a utilização da API. Esse imenso suporte que é encontrado online é decisivo para a escolha dessa plataforma de desenvolvimento, e ao longo deste trabalho exerceu um papel importantíssimo para o avanço da equipe.

Embora apresente ambiente favorável ao aprendizado, alguns problemas foram encontrados pela equipe durante este período de desenvolvimento. A maioria deles foi relacionado a testes dentro da plataforma, que se apresentaram difíceis de serem implementados para a utilização de determinados recursos da API. Houve problemas com a simulação da comunicação entre os componentes, e mesmo uma grande quantidade de esforço direcionada a esse fim ao longo de várias *sprints* não resultou em avanço nessa direção. O desenvolvimento ao longo do semestre foi bastante limitado no que diz respeito aos testes dos próprios componentes internos do aplicativo, o que é imprescindível para o desenvolvimento de um produto de qualidade. Dessa forma, uma das tarefas e provável desafio para a próxima etapa deste trabalho é realizar os testes de todos os componentes internos do aplicativo e-lastic, bem como testes de aceitação para a interface gráfica que será entregue também nesse trabalho futuro.

Embora apenas alguns testes internos tenham sido implementados com sucesso, a equipe obteve bons resultados na implementação de testes de interação com usuário, utilizando das ferramentas *calabash* [1] e *cucumber*.

[1] Para mais informações visite <http://calaba.sh/>

Devido a equipe mais inexperiente no desenvolvimento Android, grande parte do esforço neste período de trabalho foi direcionado ao treinamento da equipe e participação dentro do ciclo de desenvolvimento. Isso limitou um pouco o tempo dedicado a parte teórica de estudo e levantamento bibliográfico deste trabalho, embora tenha alcançado resultados importantes no desenvolvimento. É importante ressaltar que esse não era o objetivo inicial de trabalho, que deveria ter sido mais focado na revisão de código e no papel de consultor da plataforma em vez de implementador de fato. Com isso, houve algumas etapas com maior intervenção dentro do ciclo de desenvolvimento e outras onde a equipe levou o desenvolvimento por sua conta enquanto o papel de revisor e consultor exercido pelo principal autor deste trabalho era mantido.

Esta etapa de desenvolvimento, focada na arquitetura do aplicativo, alcançou um resultado satisfatório nos quesitos de manutenibilidade e modularização, gerando uma sólida estrutura arquitetural que será o ponto de partida para a próxima etapa deste trabalho.

Referências

ANDROIDDEVELOPERS. *Google guide to android development*. 2014. Disponível em: <http://developer.android.com/develop/index.html>. Citado na página 17.

BRAHLER, S. *Analysis of the Android Architecture*. 2010. Citado na página 26.

DEHLINGER, J.; DIXON, J. Mobile application software engineering: Challenges and research directions. 2011. Disponível em: <http://www.mobileseworkshop.org/papers/7_Dehlinger_Dixon.pdf>. Citado 2 vezes nas páginas 23 e 26.

DEVOS, M. *Bionic vs Glibc Report*. 2014. Disponível em: <http://irati.eu/wp-content/uploads/2012/07/bionic_report.pdf>. Citado na página 17.

FONSECA, C. F. *História da Computação: O caminho do pensamento e da tecnologia*. [s.n.], 2007. Disponível em: <http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf>. Citado na página 13.

GANDHEWAR, N.; SHEIKH, R. Google android: An emerging software platform for mobile devices. 2010. Disponível em: <http://www.enggjournals.com/ijcse/doc/003-IJCSESP24.pdf>. Citado na página 30.

HEUSER, S. et al. Asm: A programmable interface for extending android security. 2014. Disponível em: <http://www.icri-sc.org/fileadmin/user_upload/Group_TRUST/PubsPDF/heuser-sec14.pdf>. Citado 2 vezes nas páginas 18 e 25.

HOLZER, A.; ONDRUS, J. Trends in mobile application development. 2009. Disponível em: <http://www.janondrus.com/wp-content/uploads/2009/07/2009-Holzer-BMMP.pdf>. Citado 2 vezes nas páginas 21 e 22.

IEEE. *Guide to the Software Engineering Body of Knowledge*. 2014. Disponível em: <http://www.computer.org/portal/web/swebok/swebokv3>. Citado 5 vezes nas páginas 21, 22, 24, 27 e 28.

LOSS, J. F. et al. Quantificação da resistência oferecida por bandas elásticas. 2002. Disponível em: <https://cbce.tempsite.ws/revista/index.php/RBCE/article/download/340/296>. Citado na página 14.

MOORE, A. Mobile as 7th of the mass media: an evolving history. 2007. Disponível em: <http://smlxtralarge.com/wp-content/uploads/2008/03/smlxl-m7mm-copy.pdf>. Citado na página 13.

SCHWABER, K.; BEEDLE, M. *Agile Sofware Development with Scrum*. [S.l.: s.n.], 2002. Citado na página 33.

SHANKER, A.; LAL, S. Android porting concepts. 2011. Disponível em: <http://p0-uni.googlecode.com/svn/trunk/p0-uni/Artikler/AndroidPortingConcepts.pdf>. Citado na página 20.

SOKOLOVA, K.; LEMERCIER, M.; GARCIA, L. Android passive mvc: a novel architecture model for android application development. 2013. Disponível em: <http://www.thinkmind.org/download.php?articleid=patterns_2013_1_20_70039>. Citado na página 24.

WASSERMAN, A. I. Software engineering issues for mobile application development. 2010. Disponível em: <http://repository.cmu.edu/cgi/viewcontent.cgi?article=1040&context=silicon_valley>. Citado 3 vezes nas páginas 21, 23 e 26.