Avaliação de Plataformas Híbridas para Desenvolvimento de Aplicações para Dispositivos Móveis

Jardel dos Santos Montan[1], Marcelo Costa P Santos[1]

[1]Instituto Federal do Sudeste de Minas – Juiz de Fora – MG – Brasil

{jardel.montan@gmail.com, marcelocpsantos@gmail.com}

Abstract. *The popularity and advancement of mobile devices has made application development a challenge facing the number of existing platforms. Develop the same application for each platform may not be the best strategy. Consequently, we see the growth of the hybrid development tools, where the application is developed only once and adapted for each platform. This paper seeks to evaluate the performance of two hybrid tools in graphical interface rendering and triggering device hardware for Android platform. The results are compared to the native approach to determine how good this approach can be when compared to hybrid applications. The main goal of the paper is provide subsidies to developers in choosing the best development environment suited to their application.*

Resumo. A popularidade e avanço dos dispositivos móveis tornou o desenvolvimento de aplicativos um desafio diante do número de plataformas existente, e desenvolver o mesmo aplicativo para cada plataforma pode não ser a melhor estratégia. Consequentemente, percebemos o crescimento das ferramentas de desenvolvimento híbrido, onde o aplicativo é desenvolvido uma única vez e adaptado para cada plataforma. O presente trabalho avalia o desempenho de duas ferramentas híbridas em renderização de interface gráfica e acionamento do *hardware* do dispositivo para a plataforma Android. Os resultados são comparados com a abordagem nativa para determinar quanto melhor pode ser essa abordagem em relação as aplicações híbridas, e consequentemente, oferecer uma contribuição aos desenvolvedores na escolha do ambiente de desenvolvimento mais adequado à sua aplicação.

1. INTRODUÇÃO

Ao longo dos últimos anos presenciamos um grande avanço na tecnologia móvel. Os dispositivos móveis, em particular os *smartphones* e *tablets*, estão disponíveis em uma grande variedade de modelos e marcas, com processadores rápidos, amplo espaço de armazenamento, sensores e muitas funcionalidades. Como resultado, a utilização desses dispositivos e o mercado de aplicativos não para de crescer, as lojas apresentam uma grande variedade de tipos de aplicações para os usuários. Entretanto, o desenvolvimento de aplicativos móveis se

tornou um desafio diante da diversidade de plataformas (Android, iOS, Windows Phone, outras) que utilizam linguagens de programação e ambientes de desenvolvimento diferentes.

Hoje a forma mais direta de alcançar todas as plataformas de forma eficiente é através do desenvolvimento nativo, em que o aplicativo é desenvolvido para cada plataforma com acesso a todos os recursos do dispositivo. O desempenho nesse caso tende a ser o melhor possível, contudo, desenvolver uma versão para cada plataforma impacta em maior custo e tempo de desenvolvimento, além disso, uma alteração no aplicativo deverá ser realizada em todas as versões.

Neste contexto surge o desenvolvimento híbrido possibilitando o reaproveitamento do código na geração de aplicações para diferentes plataformas móveis, com a consequente diminuição no custo de desenvolvimento e manutenção. Existem várias ferramentas híbridas no mercado, como o Appcelerator Titanium, PhoneGap, Xamarin, entre outras. Cada ferramenta possui características diferentes e uma escolha inadequada pode gerar retrabalho e maior custo no desenvolvimento da aplicação.

O objetivo deste trabalho é mensurar quanto melhor pode ser uma aplicação nativa em relação à abordagem híbrida para aplicações que necessitam renderizar interface gráfica, acionar o *hardware* para vibrar o dispositivo e obter a localização atual. Com os resultados oferecemos subsídios para que desenvolvedores possam escolher ferramentas que melhor se adequem à sua aplicação.

2. TRABALHOS RELACIONADOS

O cenário de desenvolvimento móvel atual apresenta muitas estratégias e abordagens de desenvolvimento diferentes para quem deseja realizar estudos ou atuar nessa área. Os artigos [2], [3] e [4] contribuíram para a compreensão desse cenário pois realizam uma fragmentação das estratégias por categorias e abordagens e apresentam análises relevantes sobre cada uma.

Os artigos [2] e [4] relatam medições sobre o consumo de memória primária, secundária e capacidade de processamento dos dispositivos. [3] e [6] descrevem testes quanto à abrangência do acesso ao hardware e análises quanto a possibilidade de distribuição de aplicativos nas lojas das diversas plataformas. Os autores destacam os ganhos de tempo e custo de desenvolvimento e elegem a abordagem Tradutor de Código como a mais promissora entre as híbridas. Tradutor de Código refere-se à geração de um código compilado intermediário que é interpretado no dispositivo de execução, à semelhança do *byte-code* utilizado na linguagem Java. Essa e outras abordagens serão descritas na próxima seção.

Nosso estudo avalia o desempenho das aplicações híbridas em quesitos não avaliados diretamente nos trabalhos citados, portanto nossa contribuição acrescenta informações úteis de desempenho para os desenvolvedores e estudos futuros.

3. CATEGORIAS DE DESENVOLVIMENTO MÓVEL

3.1. Nativa

Uma aplicação nativa é desenvolvida especificamente para uma plataforma móvel utilizando ferramentas de desenvolvimento e linguagens de programação suportadas pela plataforma. Isto significa que aplicativos nativos têm total acesso aos recursos e funcionalidades do dispositivo (câmera, GPS, acelerômetro, calendário, entre outros). O desempenho, a experiência do usuário e a interface gráfica nessas aplicações tendem a ser as melhores possíveis, entretanto, desenvolver uma versão nativa para cada plataforma demanda maior custo, esforço e tempo. Uma vez desenvolvido o código para uma plataforma ele não é reutilizado para outra. A tabela 1 ilustra as tecnologias para as principais plataformas do mercado.

Plataforma	Linguagem	Ambiente de desenvolvimento
Android	Java	Android Studio / Eclipse
iOS	Objective - C	XCode
Windows Phone	C#	Visual Studio

Tabela 1 – Tecnologias para desenvolvimento nativo.

3.2. Web Apps

Estas aplicações são sites desenvolvidos com linguagens de programação web (HTML, CSS e JavaScript) e otimizados para funcionar como um aplicativo. A aplicação é executada em um *browser* do dispositivo [2][3][4][6], o que a torna compatível com todas as plataformas e de fácil atualização. O comportamento em cada plataforma depende da maneira como o *browser* processa a aplicação.

Web Apps apresenta algumas desvantagens em comparação com aplicações nativas. O acesso aos recursos do dispositivo é limitado [2][3], a experiência do usuário é afetada devido a interface não ser nativa, a aplicação não é instalada no dispositivo e geralmente só funciona com o dispositivo conectado à internet [2][4][6].

3.3. Híbrida

Aplicações híbridas são desenvolvidas uma vez e o código é compartilhado entre as plataformas. Os aplicativos podem ser publicados nas lojas e instalados no dispositivo. Há três abordagens de desenvolvimento híbrido: Tempo de Execução, Tradutor de Código e Web-para-Nativo [2] [3] [4] [6].

3.3.1. Tempo de Execução (*Runtime*)

Em abordagens Tempo de Execução o aplicativo é desenvolvido com linguagem script (JavaScript, Lua ou Rubi). A ferramenta de desenvolvimento cria um pacote de instalação do aplicativo com o interpretador da linguagem utilizada e o código script [2][3][4], sem nenhuma modificação. Em tempo de execução, o interpretador traduz o código script para código de máquina (nativo) para ser executado [2]. As ferramentas mais populares dessa abordagem são Titanium Appcelerator [15], Adobe AIR [16] e Adobe Flex [17].

3.3.2. Tradutor de Código (*Source Code Translators*)

Esta abordagem é semelhante a anterior, com duas diferenças. A primeira está na linguagem utilizada para desenvolvimento, não se limitando as linguagens script. A segunda é na compilação. O código fonte é compilado para b*yte-code* ou código de máquina (nativo) [2]. *Byte-code* é universal existindo interpretadores para as principais plataformas. Xamarin [12], QT [18] e Delphi XE6 [20] são as ferramentas mais populares dessa abordagem.

3.3.3. Web-para-Nativo (*Web-To-Native Wrapper*)

Esta abordagem mescla as vantagens das categorias Web Apps e Nativa. São utilizadas linguagens de desenvolvimento web (HTML, CSS e JavaScript), porém os aplicativos não são executados por um *browser* [2]. A aplicação é executada por um container nativo (WebView no Android e UIWebView no iOS) e o acesso ao *hardware* e recursos do dispositivo é feito por bibliotecas [6]. Para obter uma interface e experiência do usuário mais próxima da nativa, ferramentas complementares podem ser utilizadas, como Sencha Tooch [14] ou Jquery Mobile [20].

As principais desvantagens são a experiência do usuário afetada por não utilizar componentes nativos e o desempenho prejudicado pela execução no container. Cordova/PhoneGap [13] é a ferramenta mais popular.

A tabela 2 apresenta as vantagens e desvantagens entre as categorias nativa, web apps e híbrida. As aplicações híbridas se apresentam como solução mais interessante para desenvolvimento, por atender relativamente bem aos principais requisitos.

Critério	Nativa	Web Apps	Híbrida
Multiplataforma	Não	Sim	Sim
Custo de desenvolvimento	Alto	Baixo	Baixo/Médio
Desempenho	Alto	Baixo	Intermediário
Acesso ao dispositivo	Total	Limitado	Total
Utilização offline	Sim	Não	Sim
Loja de aplicativos	Sim	Não	Sim
Experiência do usuário	Alta	Baixa	Média/Alta
Facilidade de manutenção	Baixa	Alta	Alta

Tabela 2 – Vantagens e desvantagens das categorias de desenvolvimento.

5. METODOLOGIA

5.1. Plataforma Utilizada

Como podemos observar pela figura 1, que apresenta dados de novembro de 2016 levantados pelo IDC (*International Data Corporation*) [8], a plataforma Android é a que possui a maior participação no mercado, portanto, nos concentramos nela para realização dos testes e comparações.

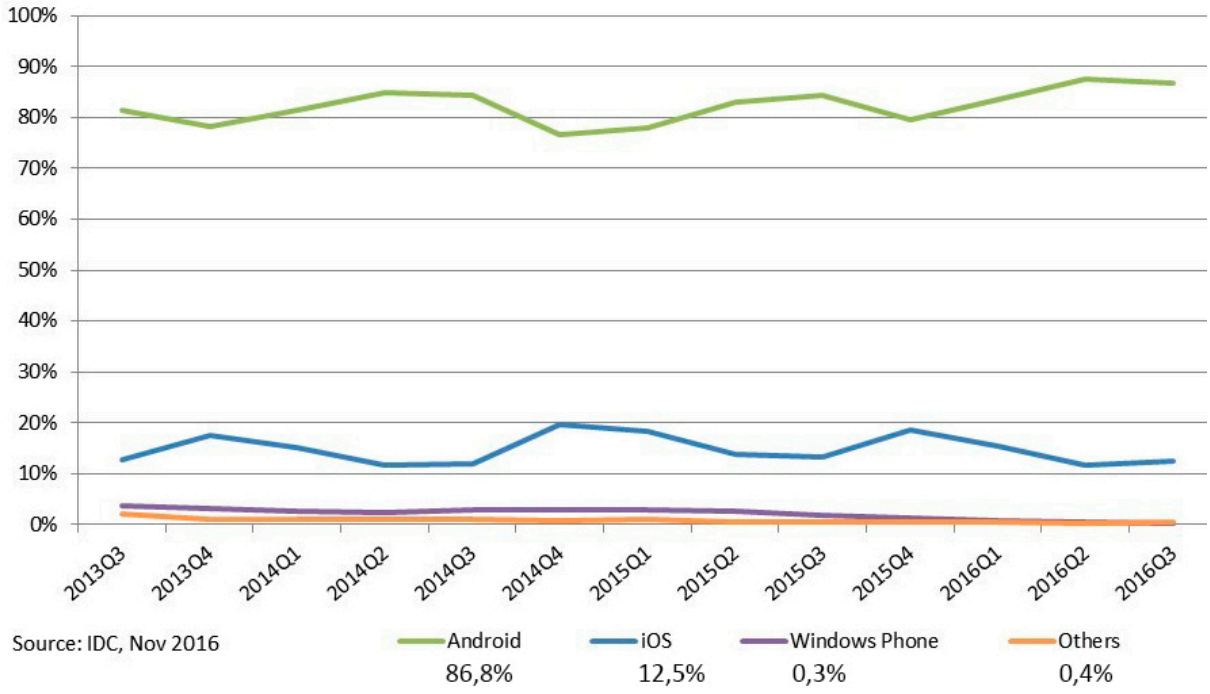

Figura 1 – Gráfico das cotas de mercado das plataformas

Fonte: IDC (2016)

5.2. Seleção das Abordagens

Para o estudo, selecionamos ferramentas de duas abordagens. Como critério, priorizamos abordagens onde os aplicativos gerados estejam o mais próximo possível de um nativo em desempenho, acesso ao dispositivo e suporte a aplicações de baixa e média complexidade.

As Web Apps não se encaixam no que tradicionalmente consideramos uma aplicação para celular, portanto, não as consideraremos nesse estudo.

Tempo de Execução e Tradutor de Código são semelhantes no sentido de incorporar na aplicação um interpretador para uma linguagem, sendo que o último utiliza b*yte-code* ao invés de uma linguagem de alto nível. O b*yte-code* tende a apresentar desempenho superior em função das otimizações conseguidas na geração do código intermediário, portanto, Tradutor de Código foi a primeira abordagem selecionada.

Web-para-Nativo é a segunda selecionada. A abordagem utiliza linguagens para desenvolvimento web, entretanto consegue acessar o dispositivo por meio de um container nativo. Além disso, pode utilizar ferramentas complementares para tornar o aplicativo mais próximo de uma aplicação nativa.

5.3. Ferramentas Selecionadas

Principalmente em função de suas grandes popularidades o Xamarin (Tradutor de Código) e PhoneGap (Web-para-Nativo) foram as ferramentas selecionadas.

Xamarin é uma ferramenta de desenvolvimento que utiliza a linguagem C# para Android, iOS e Windows Phone, compartilhando o código entre os aplicativos. Adquirida pela Microsoft 2016, tornou-se gratuita para equipes de até cinco pessoas e foi incorporada ao IDE Visual Studio possibilitando aos desenvolvedores e empresas usufruir das tecnologias .NET na criação de suas aplicações. Segundo o site da ferramenta, o reaproveitamento de código é de aproximadamente 85% para a maioria das aplicações, esse número pode variar para mais ou menos de acordo com a metodologia utilizada e tipo de aplicação, pois a ferramenta possibilita trabalhar com formas diferentes de desenvolvimento que ficam a critério do desenvolvedor.

No Android, o Xamarin compila o código para uma linguagem intermediária, e na inicialização do aplicativo o compilador *Just-in-Time* (JIT) [22] compila para montagem do

aplicativo nativo. Para o iOS, o Xamarin utiliza o *Ahead-of-Time* (AOT) [23] compilando diretamente para código de montagem nativo [2].

Phonegap é uma ferramenta de código aberto e gratuita, a mais conhecida no cenário de desenvolvimento móvel. A ferramenta possibilita o desenvolvimento das aplicações utilizando linguagens web (HTML, CSS e JavaScript), sendo uma ótima opção para desenvolvedores com conhecimento prévio dessas linguagens. Os desenvolvedores podem escolher a IDE de desenvolvimento web que melhor lhe atenda. O compartilhamento de código é de 100% e suporta as plataformas Android, iOS, Windows Phone, BlackBerry e Amazon-Fire. As aplicações geradas não possuem interface nativa, porém utilizam a tecnologia Apache Cordova para acionar recursos nativo do dispositivo. A geração dos aplicativos podem ser realizadas por um serviço na nuvem conhecido como Phonegap Build.

5.4. Aplicações

Para cada cenário (nativo, Xamarin, PhoneGap), desenvolvemos três aplicações, uma para renderizar componentes gráficos e duas para acionar *hardware* do dispositivo. Todos os códigos fontes e aplicativos estão disponíveis no endereço [21].

A avaliação do desempenho para renderização de componentes gráficos é extremamente importante pois a interface com o usuário baseia-se fortemente no *display* do dispositivo. A medição do desempenho nas diferentes plataformas de maneira justa é um desafio pois cada abordagem é diferente na forma de tratar o problema. A Web-para-Nativo (PhoneGap) é baseada em html, e a renderização é feita utilizando o mesmo software dos navegadores e a Tradutor de Código (Xamarin) os componentes de interface são nativos. Apesar das diferenças inerentes de cada plataforma, tentamos construir os testes de forma mais justa possível. As telas construídas podem ser observadas na Figura 2.

O acesso ao *hardware* é outro ponto importante para determinados tipos de aplicações e frequentemente citado como a principal razão para o desenvolvimento de aplicações nativas em detrimento de plataformas híbridas de desenvolvimento.

Na aplicação responsável por renderizar componentes gráficos incluímos caixas de texto, caixas de seleção, botões de rádio, tabelas, imagens, botões e texto. A interface foi programada para se comportar como um aplicativo comum quanto à organização dos elementos e na utilização de cores. A Figura 2 mostra, a título de exemplo, duas partes da tela renderizada nos testes. A tela completa é grande, ocupando aproximadamente nove vezes o tamanho da tela do aparelho utilizado nos testes.

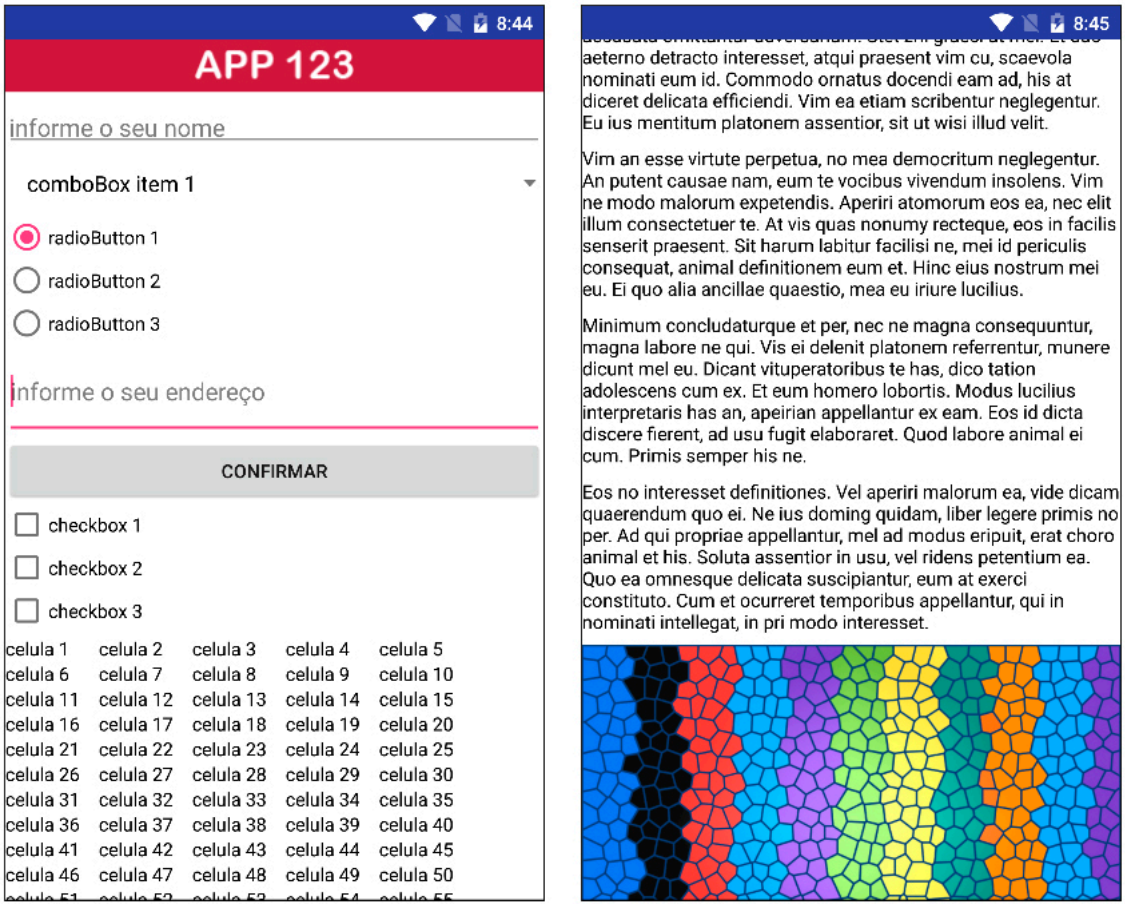

Figura 2 – Aplicativo de renderização de interface gráfica.

As aplicações responsáveis por acionar o *hardware* do dispositivo contam com um botão que realiza a requisição quando pressionado. Uma aplicação solicita que o dispositivo vibre por dois segundos e a outra solicita a localização (latitude e longitude) atual do dispositivo via GPS. Utilizamos uma opção suportada pela plataforma Android chamada *HighAccuracy* para que a precisão na localização seja a melhor possível para o local onde o dispositivo se encontra. A figura 3 mostra a tela dos dois aplicativos.

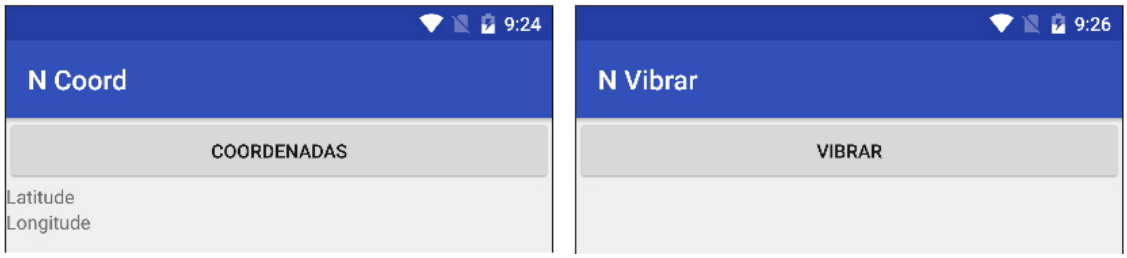

Figura 3 – Aplicativos responsáveis por acionar o *hardware* do dispositivo.

5.5. Testes

Os testes foram realizados com nível de bateria sempre maior que 75%, com o aparelho em "modo avião", rotação automática desabilitada e a localização configurada no modo de alta precisão. Antes de cada teste ser iniciado o dispositivo foi reiniciado e a cada medição o aplicativo foi fechado e removido da memória, ou seja, para dez medições o aplicativo foi aberto dez vezes. A tabela 3 fornece as especificações do dispositivo utilizado para as medições.

Dispositivo	Sistema Operacional	Memória RAM	Processador
Moto G 1º Geração	Android 5.1	1 GB	1.2 GHz Quad-Core

Tabela 3 – Especificações do dispositivo.

Durante as medições dos aplicativos de renderização de interface gráfica verificamos que a aplicação desenvolvida no PhoneGap se comportava de forma diferente: a notificação inserida para exibir o tempo total ao final da renderização era exibida antes da renderização dos componentes. O sistema adia a renderização do documento principal da aplicação, dando preferência à janela de diálogo que informa o tempo de processamento ficando a medição, portanto, totalmente comprometida. Por este motivo optamos por não utilizar um botão responsável por acionar o início da renderização para essas aplicações, em vez disso, a renderização inicia junto com a aplicação.

Com o abandono do botão, o registro do tempo inicial se dá com a carga da aplicação, o que poderia contaminar o tempo total medido com o tempo de carga do aplicativo. A fim de contornar o problema foram medidos os tempos de renderização de uma tela vazia e os valores obtidos subtraídos dos tempos de renderização de tela cheia garantindo, portanto, que os tempos obtidos refiram-se exclusivamente à renderização dos componentes da interface gráfica do aplicativo. As médias dos tempos de renderização, em dez medições, das telas cheia e vazia podem ser observados na tabela 4.

Para as aplicações que acionam o *hardware* do dispositivo e contam com um botão, a medição inicia quando o botão é pressionado e termina quando já tenhamos obtido a resposta ou já tenha sido acionado o *hardware* em questão.

Nas aplicações nativas a classe utilizada para captura dos tempos foi a *System.nanoTime* [9] da linguagem Java. Para as aplicações no Xamarin, foi utilizada a classe *System.Diagnostics.Stopwatch* [10] disponível no C#. Já nas aplicações PhoneGap que

utilizam JavaScript, as medições foram realizadas com a API (*Application Programming Interface*) *performance* [11].

6. RESULTADOS

Foram realizadas dez medições para cada aplicação e para melhor compreensão, apenas as médias e os desvios são apresentados nos resultados. As análises e discussões são tratadas com maior ênfase na sessão 7.

6.1. Aplicativo de Renderização de Interface Gráfica

A tabela 4 contém os resultados para os aplicativos de tela cheia e tela vazia, conforme explicado na sessão anterior. A aplicação desenvolvida com PhoneGap apresentou um desempenho bastante eficiente.

	Tela Cheia		Tela Vazia		
Ferramenta	**Desvio**	**Média**	**Desvio**	**Média**	**Média Total**
Nativo	0,70	263,90	6,90	5,50	258,40
PhoneGap	10,50	387,20	43,12	112,50	274,70
Xamarin	0,18	436,60	6,12	4,10	432,50

Tabela 4 – Tempos para renderização (em milissegundos).

6.2. Aplicativo para Vibrar o Dispositivo

Xamarin e PhoneGap apresentaram um desempenho semelhante nesse requisito. Os resultados podem ser vistos na tabela 5.

Ferramenta	**Desvio**	**Média**
Nativo	0,32	2,20
PhoneGap	6,00	12,50
Xamarin	2,52	10,60

Tabela 5 – Tempos para vibrar (em milissegundos).

6.3. Aplicativo para Obter a Localização do Dispositivo

Esta avaliação foi a que apresentou maior diferença no desempenho, o PhoneGap precisou do dobro do tempo médio do Xamarin. Os resultados são apresentados na tabela 6.

Ferramenta	**Desvio**	**Média**
Nativo	8,92	25,90
PhoneGap	425,04	814,20

Xamarin	18,90	64,10

Tabela 6 – Tempos para obter a localização (em milissegundos).

7. CONCLUSÃO

Este trabalho apresentou uma análise de desempenho de aplicações móveis em renderização de interface gráfica e acesso ao *hardware* do dispositivo utilizando duas ferramentas de desenvolvimento móvel multiplataforma, Xamarin e PhoneGap. Cada ferramenta representa uma abordagem de desenvolvimento móvel diferente. Os resultados obtidos foram comparados com o desenvolvimento nativo para determinar quanto cada ferramenta perde em desempenho.

Em renderização de interface o PhoneGap obteve desempenho semelhante ao nativo e 55% mais eficiente que o Xamarin. A explicação para o bom desempenho pode estar na WebView onde a aplicação é executada, esse recurso da plataforma Android é utilizado por grandes empresas e fabricantes, dessa forma, a renderização do código web pode ser bastante otimizada. Além disso, a interface gráfica gerada pelo PhoneGap não é nativa e isto pode se traduzir em um ganho de desempenho.

No acesso ao *hardware*, para fazer o dispositivo vibrar as duas ferramentas alcançaram desempenho semelhante e bom em relação à aplicação nativa. Na aplicação para obter a localização com GPS, o PhoneGap apresentou um desempenho bastante desfavorável em relação tanto a aplicação nativa quanto ao Xamarin.

Durante todas as medições as aplicações desenvolvidas com PhoneGap, apresentaram desvio padrão mais alto, enquanto que as aplicações desenvolvidas com Xamarin foram mais homogêneas. Associamos essa observação ao fato do PhoneGap utilizar componentes externos (WebView, bibliotecas), ficando mais vulnerável às políticas de escalonamento dos sistemas operacionais.

Para aplicações em que o desempenho e/ou interface gráfica nativa são critérios, o Xamarin é a ferramenta mais recomendada. Quando esses critérios não são requisitos o PhoneGap se torna uma boa opção, pois o reaproveitando de código é um pouco maior e a utilização de ferramentas auxiliares podem reduzir o tempo de desenvolvimento da aplicação.

8. Trabalhos Futuros

As conclusões de desempenho obtidas foram bastante satisfatórias para a plataforma Android, entretanto é importante realizar os testes e análises para as plataformas iOS e Windows Phone

a fim de oferecer uma contribuição mais ampla para os desenvolvedores. Pretende-se também incluir novas ferramentas e avaliações, como por exemplo, o acesso à câmera do dispositivo e a um banco de dados local para leitura e escrita.

9. Referências

[1] Boushehrinejadmoradi, N., Ganapathy, V. & Nagarakatte, S. Testing Cross-Platform Mobile App Development Frameworks. Automated Software Engineering (ASE), 30th International Conference on. IEEE/ACM, 2015, p. 441-451.

[2] Willocx, M., Vossaert, J., & Naessens, V. A Quantitative Assessment of Performance in Mobile App Development Tools. International Conference on Mobile Services. IEEE, 2015, p. 454-461.

[3] Delia, L., Galdamez, N., Thomas, P., et al. Multi-platform mobile application development analysis. 9th International Conference on Research Challenges in Information Science (RCIS). IEEE, 2015, p. 181-186.

[4] Charkaoui, S., Adraoui, Z., & Benlahmar, E. H. Cross-platform mobile development approaches. Third IEEE International Colloquium in Information Science and Technology (CIST), IEEE, 2014, p. 188-191.

[5] Dalmasso, I., Datta, S. K., Bonnet, C., & Nikaein, N. Survey, comparison and evaluation of cross platform mobile application development tools. 9th International Wireless Communications and Mobile Computing Conference (IWCMC), IEEE, 2013, p. 323-328.

[6] Xanthopoulos, S., & Xinogalos, S. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. Proceedings of the 6th Balkan Conference in Informatics. ACM, 2013, p. 213-220.

[7] Hui, N. M., Chieng, L. B., Ting, W. Y., et al. Cross-platform mobile applications for android and iOS. Wireless and Mobile Networking Conference (WMNC), 6th Joint IFIP. IEEE, 2013, p. 1-4.

[8] International Data Corporation (IDC) - Smartphone os market share, q3 2016. Disponível em: <http://www.idc.com/promo/smartphone-market-share/os>. Acesso em dezembro, 2016.

[9] Oracle – Classe System.nanoTime da linguagem Java. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime()>. Acesso em outubro, 2016.

[10] Microsoft – Classe System.Diagnostics.StopWatch da linguagem C#. Disponível em: <https://msdn.microsoft.com/pt-br/library/system.diagnostics.stopwatch(v=vs.110).aspx>. Acesso em outubro, 2016.

[11] Mozzila – API Performance da linguagem JavaScript. Disponível em:

[12] Xamarin. Disponível em: <https://www.xamarin.com/> Acesso em agosto, 2016.

[13] PhoneGap. Disponível em: <http://phonegap.com/> Acesso em agosto, 2016.

[14] Sencha. Disponível em: <https://www.sencha.com/> Acesso em agosto, 2016.

[15] Titanium Appcelerator. Disponível em: <http://www.appcelerator.com/> Acesso em agosto, 2016.

[16] Adobe AIR. Disponível em: <https://get.adobe.com/br/air/> Acesso em agosto, 2016.

[17] Adobe Flex. Disponível em: <http://www.adobe.com/br/products/flex.html> Acesso em agosto, 2016.

[18] QT. Disponível em: <https://www.qt.io/qt-for-application-development/> Acesso em agosto, 2016.

[19] Delphi XE6. Disponível em: <https://www.embarcadero.com/products/delphi> Acesso em agosto, 2016.

[20] Jquery Mobile. Disponível em: <https://www.embarcadero.com/products/delphi> Acesso em agosto, 2016.

[21] Aplicações e códigos do trabalho. Disponível em: <https://drive.google.com/open?id=0B5jZ02_lHbHuaHJ3bWtSUmtWMm8>.

[22] Aycock, John. A Brief History of Just-In-Time. ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 97–113.

[23] Hong, SungHyun and Kim, et al. ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 97–113.