AVALIAÇÃO DE PLATAFORMAS HÍBRIDAS PARA DESENVOLVIMENTO DE APLICAÇÕES PARA O ANDROID

Jardel dos Santos Montan[1], Marcelo Costa Pinto e Santos[2]

RESUMO: A popularidade e avanço dos dispositivos móveis tornou o desenvolvimento de aplicativos um desafio diante do número de plataformas existentes. 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 tempo de processamento de duas ferramentas híbridas, o Xamarin e o PhoneGap, em renderização de interface gráfica e acionamento do hardware do dispositivo para a plataforma Android. Os resultados são comparados com o desenvolvimento nativo a fim de subsidiar desenvolvedores na escolha do ambiente de desenvolvimento mais adequado à sua aplicação.

PALAVRAS-CHAVE: Aplicações Móveis, Multiplataforma.

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 para eles não para de crescer e as lojas apresentam uma grande variedade de tipos de aplicações para os usuários.

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. Do ponto de vista do desempenho, a melhor forma de desenvolver um aplicativo para um dispositivo móvel é a utilização direta dos recursos oferecidos pelos desenvolvedores do dispositivo (**desenvolvimento nativo**) contudo, criar uma versão para cada plataforma aumenta o custo de desenvolvimento e manutenção. Tal aumento de custo é mantido durante toda a vida do sistema, sempre que uma nova versão for disponibilizada.

Neste contexto surge a possibilidade do **desenvolvimento híbrido** que possibilita 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 (ANDERSON, 2013), PhoneGap (GHATOL; PATEL, 2012), Xamarin (XAMARIN, 2017),

[1]Jardel dos Santos Montan, IF Sudeste MG - Campus Juiz de Fora
[2]Marcelo Costa Pinto e Santos, IF Sudeste MG - Campus Juiz de Fora, marcelocpsantos@gmail.com

entre outras. Cada ferramenta possui características diferentes e uma escolha inadequada pode gerar retrabalho e maior custo no desenvolvimento da aplicação.

O objetivo desse trabalho é avaliar quanto melhor pode ser uma aplicação nativa em relação à uma abordagem híbrida para aplicações que necessitam renderizar interface gráfica, acionar o hardware para vibrar e obter a localização (latitude e longitude) do dispositivo. As principais ferramentas existentes atualmente no mercado foram classificadas e selecionou-se o principal representante de cada categoria para testes: o Xamarin (XAMARIN, 2017) e o PhoneGap (GHATOL; PATEL, 2012). Essas plataformas foram avaliadas quanto ao tempo para renderização de tela e acesso a hardware (Vibracall e GPS). Com os resultados oferecemos subsídios para que desenvolvedores possam escolher ferramentas que melhor se adéquem à sua aplicação.

2.TRABALHOS RELACIONADOS

Vários artigos contribuem para o entendimento do atual cenário do desenvolvimento para dispositivos móveis na atualidade.

(CHARKAOUI et al., 2014) classificam e comparam qualitativamente diferentes abordagens para desenvolvimento multiplataforma, sem avaliações objetivas.

(WILLOCX; VOSSAERT; e NAESSENS, 2015) testam as mesmas ferramentas para desenvolvimento híbrido utilizadas nesse artigo quanto ao tempo para início, tempo para pausa e retomada, tempo para troca de páginas, consumo de memória, CPU, disco e bateria.

(DELIA et al., 2015) avaliam diferentes abordagens para o desenvolvimento híbrido de uma aplicação específica: um ambiente virtual de ensino. Foram testados diferentes ambientes operacionais (Android, iOS, Windows Phone, Amazon Fire OS). Realizam testes subjetivos como a experiência do usuário e facilidade para desenvolvimento.

(XANTHOPOULOS; XINOGALOS, 2013) avaliam aplicações criadas por ambientes com diferentes abordagens para o desenvolvimento híbrido. Usam métricas bastante subjetivas como a facilidade de distribuição nas lojas de cada ambiente operacional, popularidade da ferramenta, abrangência do acesso ao hardware, qualidade da interface e desempenho percebidos pelo usuário.

3. CATEGORIAS DE AMBIENTES PARA DESENVOLVIMENTO MÓVEL

Nessa sessão, apresentamos as características das principais plataformas disponíveis no mercado. Essas características serão utilizadas posteriormente na seleção dos ambientes que serão avaliados mais detidamente na seção que a segue.

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 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.

3.2. WEB APPS

Essas aplicações são sites desenvolvidos com tecnologias voltadas para a WWW (HTML, CSS e JavaScript) e otimizados para funcionar como um aplicativo. A aplicação é executada em um navegador do dispositivo, o que a torna compatível

com todas as plataformas e de fácil atualização. O comportamento em cada plataforma depende da maneira como o navegador processa a aplicação.

Web Apps apresentam algumas desvantagens em comparação com aplicações nativas. O acesso aos recursos do dispositivo é limitado e a aplicação não é instalada no dispositivo e geralmente só funciona com o dispositivo conectado à internet.

3.3 HÍBRIDA

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

Na abordagem **Tempo de Execução** o aplicativo é desenvolvido com linguagem script (JavaScript, Lua ou Ruby). A ferramenta de desenvolvimento cria um pacote de instalação do aplicativo com o interpretador da linguagem utilizada e o código script, sem nenhuma modificação. Em tempo de execução, o interpretador traduz o código script para código de máquina (nativo) para ser executado. Titanium Appcelerator (ANDERSON, 2013), Adobe AIR (LU; ZHANG, 2013) e Adobe Flex (BALDERSON et al., 2011) são as ferramentas mais populares dessa abordagem.

Tradutores de Código são semelhantes à abordagem 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 byte-code ou código de máquina (nativo). Byte-code é universal existindo interpretadores para as principais plataformas. Xamarin (XAMARIN, 2017), QT (QT, 2017) e Delphi XE6 (EMBARCADERO, 2017) são as ferramentas mais populares dessa abordagem.

A abordagem **Web-para-Nativo** (Web-To-Native Wrapper) mescla as vantagens das Web Apps com aplicações nativas. São utilizadas linguagens de desenvolvimento WEB (HTML, CSS e JavaScript), porém os aplicativos não são executados por um navegador. 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. Para obter uma interface mais próxima da nativa, ferramentas complementares podem ser utilizadas, como Sencha Tooch (SENCHA, 2017) ou Jquery Mobile (JQUERY, 2017). As principais desvantagens devem-se a interface ser distinta da interface nativa da plataforma, já conhecida pelo usuário, e o desempenho, prejudicado pela execução no container. Cordova/PhoneGap (GHATOL; PATEL, 2012) é a ferramenta mais popular que emprega essa abordagem.

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.

Tabela 2 - Vantagens e desvantagens das categorias de desenvolvimento.

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

METODOLOGIA

4.1 PLATAFORMA UTILIZADA

Como podemos observar na Figura 1 (IDC, 2016), a plataforma Android é a que possui a maior participação no mercado, portanto, nos concentramos nela para realização dos testes e comparações.

Tabela 1 - Tecnologias para desenvolvimento nativo

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

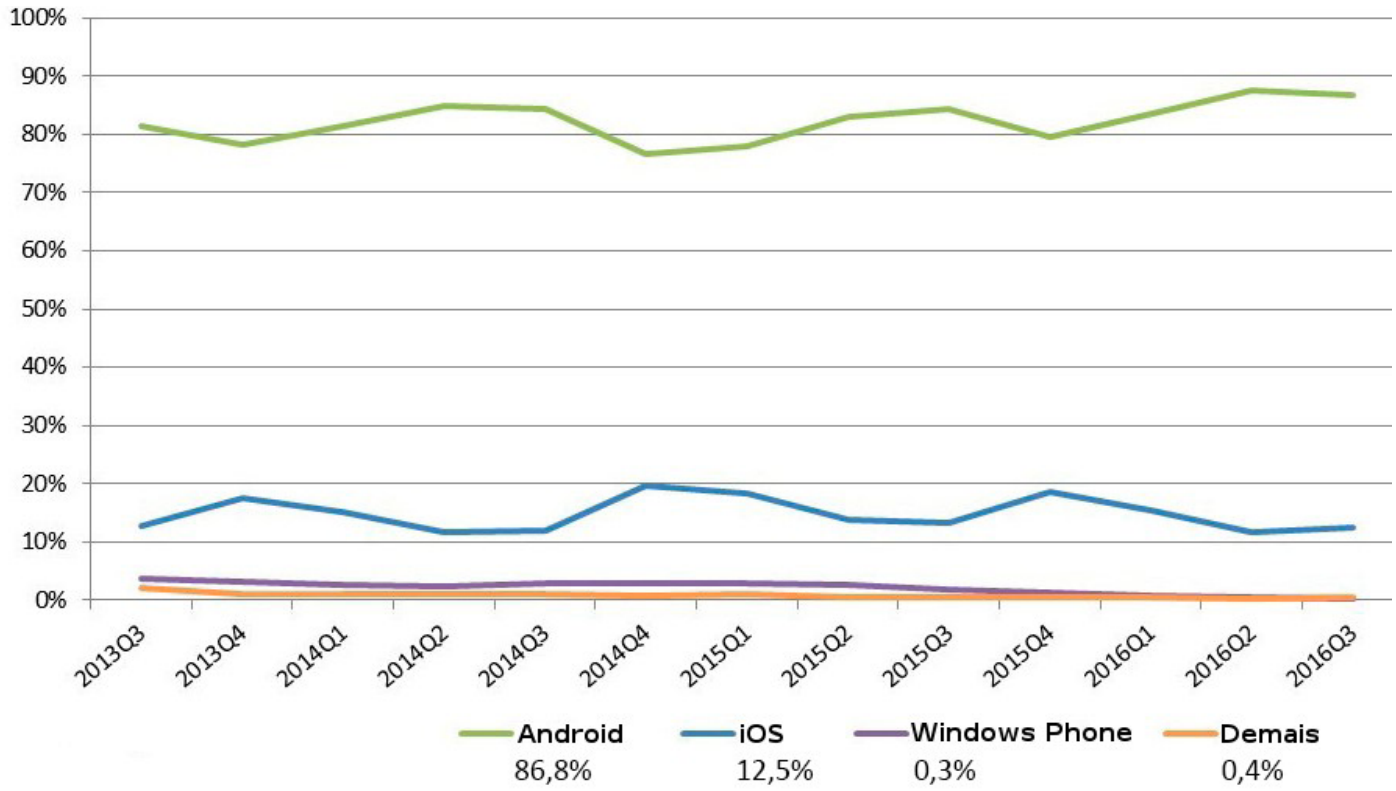

Figura 1 - Demonstrativo das cotas de mercado das diferentes plataformas (IDC, 2016).

4.2. 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 byte-code ao invés de uma linguagem de alto nível. O byte-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.

4.3. FERRAMENTAS SELECIONADAS

Principalmente em função de suas grandes popularidades (WILLOCX; VOSSAERT; e NAESSENS, 2015) 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) (AYCOCK, 2003) compila para montagem do aplicativo nativo. Para o iOS, o Xamarin utiliza o Ahead-of-Time (AOT) (HONG et al., 2007) compilando diretamente para código de montagem nativo (WILLOCX; VOSSAERT; e NAESSENS, 2015).

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 praticamente 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.

4.4. APLICAÇÕES

Para cada cenário (Nativo, Xamarin e PhoneGap), desenvolvemos três aplicações, uma para renderizar componentes gráficos e duas para acionar hardware do dispositivo, um atuador (vibracall) e um sensor (GPS). Todos os códigos fontes e aplicativos estão disponíveis no endereço (MONTAN, 2017).

A avaliação do desempenho para renderização de componentes gráficos é extremamente importante pois a interface com o usuário baseia-se fortemente na tela 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.

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.

Durante as medições dos aplicativos de renderização de interface gráfica verificamos que a aplicação desenvolvida com o PhoneGap se comportava de maneira pe-

culiar. Um botão era acionado registrando o tempo inicial e ao final dos comandos que ¬constroem os elementos de interface, o tempo final registrado e uma janela popup exibia o tempo de renderização (final menos inicial). No PhoneGap, pode-se observar que a janela popup com o resultado era exibida sobre uma tela vazia. Somente após fecharmos a janela popup os elementos de interface apareciam na janela principal. O WebView adia a construção dos elementos de interface, priorizando o popup e comprometendo as medições.

O problema foi contornado construindo-se a tela independentemente de botões e janelas popup, automaticamente quando a aplicação é carregada, e medindo os tempos inicial e final de carga da aplicação. Para expurgar tempos não relacionados à renderização, foram construídas duas aplicações para cada ambiente, uma renderizando uma tela vazia e outra renderizando a tela cheia exibida na Figura 2. O tempo considerado foi a subtração do tempo observado com a tela vazia do tempo observado com tela cheia. Dessa forma registramos o tempo gasto exclusivamente na renderização da tela.

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.
As aplicações responsáveis por acionar o hardware 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 enableHighAccuracy (MOZZILA, 2017) para que a precisão na localização

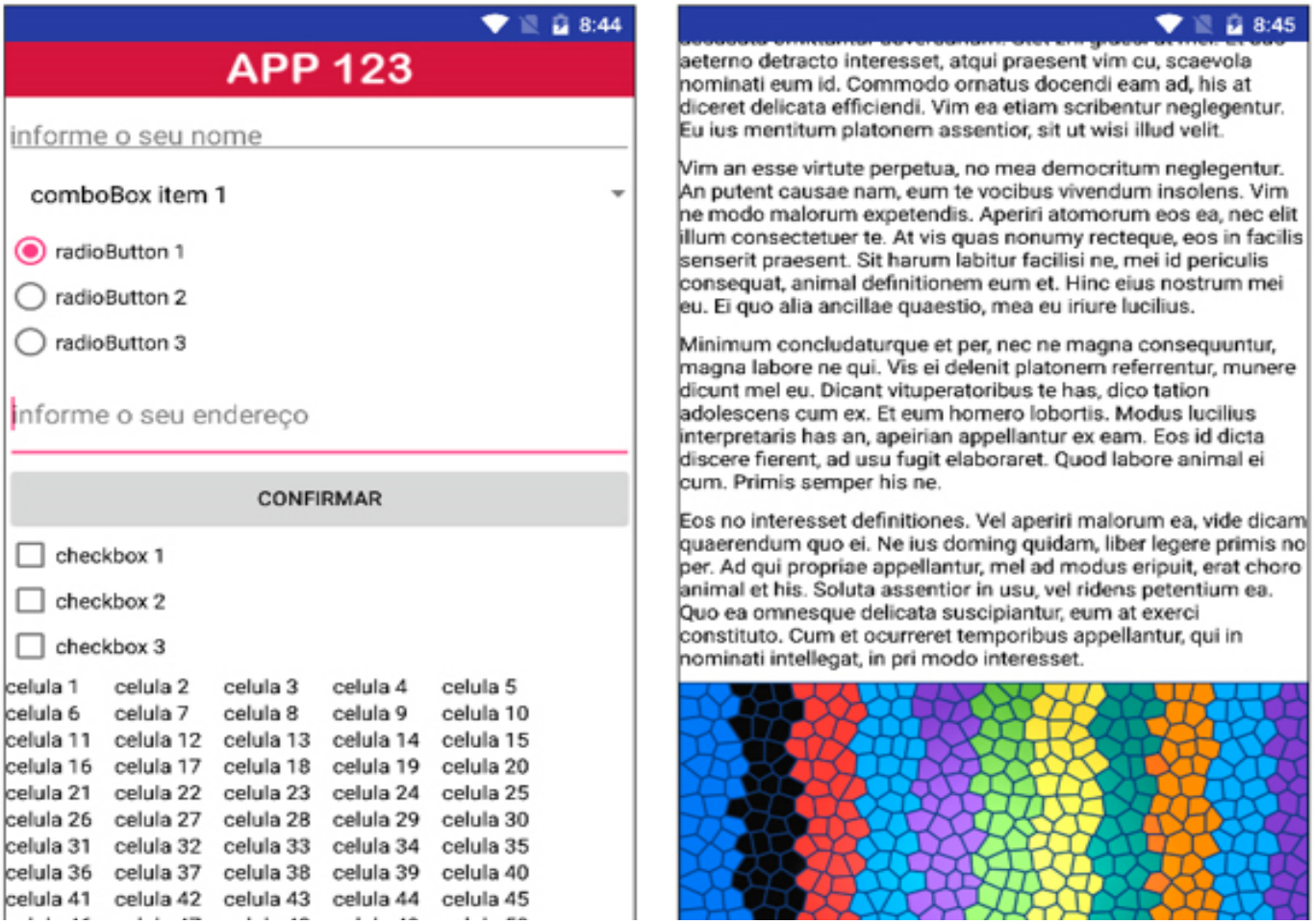

Figura 2 - Telas utilizadas para medição de desempenho de renderização de componentes gráficos.

seja a melhor possível. A Figura 3 mostra a tela dos dois aplicativos.

0Para 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 terminado o acionamento do hardware em questão.

Nas aplicações nativas a classe utilizada para captura dos tempos foi a System.nanoTime (ORACLE, 2017) para Java. Para as aplicações no Xamarin, foi utilizada a classe System.Diagnostics.Stopwatch (MICROSOFT, 2017) para C#. Nas aplicações PhoneGap, construídas com JavaScript, as medições foram realizadas com a API performance (MOZZILA, 2017).

5. RESULTADOS OBSERVADOS

Os testes foram realizados em um Motorola Moto G primeira geração, com Android 5.1, 1GB de RAM e processador Quad-Core de 1.2GHz. O nível de bateria foi mantido 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. As médias referem-se sempre a dez experiências.A Tabela 3 contém os resultados para os aplicativos de tela cheia e tela vazia. Destacamos o bom desempenho da aplicação construída com a utilização do PhoneGap. Atribuimos esse resultado ao

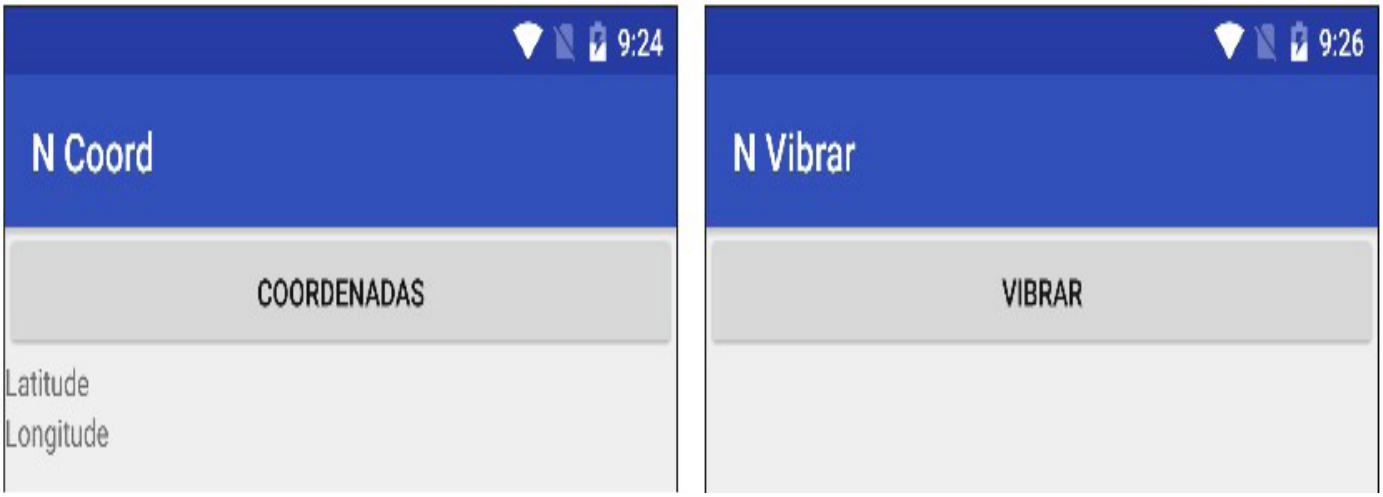

Figura 3 - Aplicativos responsáveis por acionar o hardware do dispositivo.

alto desempenho do WebView, aplicação utilizada por diversos navegadores do Android e, portanto, desenvolvida, testada, depurada e otimizada com muito interesse por várias equipes de desenvolvedores.

Na Tabela 4 podemos observar os tempos de acesso ao vibracall. Como esperado, os tempos das aplicações hibridas são consideravelmente superiores aos da aplicação nativa. PhoneGap e Xamarin apresentaram resultados bastante semelhantes.

Tabela 3 - Tempos para renderização de interface gráfica (em milissegundos).

Ferramenta	Tela Cheia		Tela Vazia		Tempo Considerado
	Desvio	Média	Desvio	Média	
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

Na Tabela 5 podemos observar os tempos para acesso ao GPS. Destaque para o desempenho muito superior do Xamarin em comparação ao PhoneGap.

Tabela 4 - Tempos para acionamento do vibracall (milissegundos).

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

Tabela 5 - Tempos para obter a localização com GPS (milissegundos).

Ferramenta	Desvio	Média
Nativo	8,92	25,90
PhoneGap	425,04	814,20
Xamarin	18,90	64,10

Destacamos o desvio padrão consideravelmente mais elevado no PhoneGap. Atribuímos o fato à utilização do WebView pelo PhoneGap. Com o envolvimento de mais de um processo para a realização da tarefa, o tempo final recebe maior influência do escalonador, consequentemente aumentando a aleatoriedade na execução.

6. 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, no sistema operacional mais utilizado atualmente. As duas ferramentas selecionadas são as mais utilizadas entre as disponíveis para as duas abordagens mais promissoras para geração de aplicações híbridas. Os resultados obtidos foram comparados com o desenvolvimento nativo.

Como poderíamos esperar, as abordagens hibridas sempre apresentaram desempenho inferior ao desenvolvimento nativo. Podemos destacar a diferença pequena entre as ferramentas híbridas e o desenvolvimento nativo para renderização de telas (6% para o PhoneGap e 67% para o Xamarin).

Para acionamento do hardware (vibracall e GPS) as diferenças foram bem mais significativas (aplicações hibridas entre 5 e 91 vezes mais lentas que as nativas). Uma comparação entre os tempos de execução de aplicações hibridas e nativas podem ser observadas na Tabela 6. Principalmente em função do bom desempenho para renderização de interface gráfica, intenso em qualquer aplicação, detectamos fortes indícios de que o desenvolvimento híbrido constitui uma opção muito interessante para o desenvolvimento de aplicações para dispositivos móveis que façam uso moderado dos sensores a atuadores específicos.

O PhoneGap apresentou a maior variabilidade, tanto em diferentes repetições do mesmo experimento (desvio padrão alto) quanto a diferentes experimentos realizados (apenas 6% mais lento na renderização de tela e 91 vezes mais lento no acesso ao GPS), no entanto, tem a vantagem de produzir, além das aplicações para instalação nas diversas plataformas, uma Web App, que torna o sistema acessível via navegador, sem a instalação de qualquer aplicativo, portanto, deve ser considerada quando a velocidade de acesso ao hardware não for crucial para a aplicação.

O Xamarin apresentou desempenho mais consistente em todos os experimentos, e velocidade muito maior no acesso ao GPS, portanto, principalmente para os desenvolvedores fluentes em C#, constitui-se na melhor opção para desenvolvimento híbrido. Apesar de se tratar de um estudo inconclusivo por não realizar testes em outras plataformas, notadamente o iOS, e limitar-se a medir o desempenho da renderização de interfaces gráficas, acesso a um atuador (vibracall) e um sensor (GPS) esperamos que, juntamente com os outros trabalhos citados que também realizaram avaliações parciais, acrescente subsídio importante na definição da plataforma a ser utilizada pelos desenvolvedores em futuros projetos.

Tabela 6 - Razão entre os tempos de execução dos métodos híbridos pelo tempo de execução da aplicação nativa

Ferramenta	Renderização	*Vibracall*	GPS
PhoneGap	1,06	5,68	91,28
Xamarin	1,67	4,82	2,47

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 processing time of two hybrid tools, Xamarin and PhoneGap, in graphical interface rendering and triggering device hardware for Android platform. The results are compared to the native development to determine how good this type of development can be when compared to hybrid development. The main goal of the paper is provide subsidies to developers in choosing the best development environment suited to their application.*

KEYWORDS: *Mobile application, Cross-platform, Multi-platform.*

REFERÊNCIAS:

ANDERSON, John. **Appcelerator Titanium: Up and Running**. " O'Reilly Media, Inc.", 2013.

AYCOCK, J. **A brief history of just-in-time.** ACM Comput. Surv., 35(2):97–113. 2003.

BALDERSON, Joseph, et al. **Professional adobe flex 3**. John Wiley & Sons, 2011.

CHARKAOUI, S., ADRAOUI, Z., et al. **Cross-platform mobile development approaches.** In Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in, pages 188–191. IEEE. 2014.

DELIA, L., et al. P. **Multi-platform mobile application development analysis**. In Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on, pages 181–186. IEEE. 2015.

EMBARDADERO. **Embarcadero – Delphi.** http://www.embarcadero.com/products/delphi. Acessado: 02/06/2017. 2017.

GHATOL, Rohit, PATEL Yogesh. **Beginning Phonegap**. New York: Apress Media, 2012.

HONG, S., et al. **Java client ahead-of-time compiler for embedded systems**. In Pro-ceedings of the 2007 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES '07, pages 63–72, New York, NY, USA. ACM. 2007.

IDC. **International data corporation (idc) - smartphone os market share**, q3 2016. http://www.idc.com/promo/smartphone-market-share/os . Acessado: 02/06/2017. 2016.

JQUERY. **Jquery mobile**. http://www.jquerymobile.com/ .Acessado: 02/06/2017. 2017.

LU, JianFeng, ZHANG, Yao. **Mobile application development based on Adobe AIR**. Electronics Information and Emergency Communication (ICEIEC), 2013 IEEE 4th International Conference on. IEEE, 2013.

MICROSOFT. **Classe stopwatch**. https://msdn.microsoft.com/ptbr/library/system. diagnostics. stopwatch(v=vs.110).aspx. Acessado: 02/06/2017. 2017.

MONTAN, Jardel dos S. **Fonte das apps utilizadas nesse artigo**. https://drive. google.com/open?id=0B5jZ02_lHbHuaHJ3bWtSUmtWMm8 . Acessado: 02/06/2017. 2017.

MOZZILA. **Mozzila - api performance da linguagem javascript**. https://developer. mozilla.org/ pt-BR/docs/Web/API/ Acessado: 02/06/2017. 2017.

ORACLE. **Oracle - classe system.nanotime da linguagem java**. https://docs. oracle.com/javase/7/ docs/api/java/lang/System.html#nanoTime(). Acessado: 02/06/2017. 2017.

QT. **Qt for Application Development**, http://www.qt.io/qt-for-application-development. Acessado:02/06/2017. 2017.

SENCHA. **Build Data-Intensive Web Apps - Faster**. http://www.sencha.com. Acessado: 02/06/2017. 2017.

WILLOCX, M., VOSSAERT, J., e NAESSENS, V. **A quantitative assessment of performance in mobile app development tools. In Mobile Services (MS)**. IEEE International Conference on, pages 454–461. IEEE. 2015.

XAMARIN. **Everything you need to deliver great mobile apps**. http://www.xamarin.com/ . Acessado: 02/06/2017. 2017.

XANTHOPOULOS, S. e XINOGALOS, S. **A comparative analysis of cross-platform development approaches for mobile applications.** In Proceedings of the 6th Balkan Conference in Informatics, pages 213–220. ACM. 2013.

Submetido em: 08/06/2017
Aceito em: 30/11/2017