DEPARTAMENTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Leonardo Schmitz

# ANÁLISE DE FERRAMENTAS DE DESENVOLVIMENTO MULTIPLATAFORMA PARA CRIAÇÃO DE APLICATIVOS MÓVEIS

Santa Cruz do Sul
2016

Leonardo Schmitz

# ANÁLISE DE FERRAMENTAS DE DESENVOLVIMENTO MULTIPLATAFORMA PARA CRIAÇÃO DE APLICATIVOS MÓVEIS

Trabalho de Conclusão de Curso II apresentado ao Curso de Ciência da Computação da Universidade de Santa Cruz do Sul para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Gilson Augusto Helfer

Santa Cruz do Sul
2016

## AGRADECIMENTOS

Agradeço especialmente ao meu orientador, Prof. Me. Gilson Augusto Helfer, pela atenção, conselhos e paciência durante o desenvolvimento deste trabalho.

Aos professores Me. Daniela Saccol Peranconi, Me. Charles Varlei Neu e Me. Beatriz Lux que, juntamente com meu orientador, contribuíram na condução e andamento deste trabalho.

A todos os professores da Unisc com quem tive contato e que sempre deram o melhor de si na maravilhosa arte da troca de conhecimento.

A minha esposa Deisi J. Rodrigues, pelos incentivos, pelo amor e compreensão de minha eventual ausência para dedicação a este trabalho.

Aos meus pais Nair Schmitz e Gelcy Schmitz, que muito se esforçaram e ainda o fazem até hoje, sempre me apoiando e amando, mesmo à distância.

Aos meus sogros Lurdes O. Rodrigues e Manoel O. Rodrigues, por serem meus segundos pais e permitirem que eu seja parte de sua família.

A todos meus colegas da Unisc, pela convivência e amizade.

Aos meus colegas de trabalho da Benoit, uma equipe sem igual.

Enfim, a todos, amigos, conhecidos e desconhecidos, que apoiaram e contribuíram na conclusão deste trabalho e da minha graduação.

## RESUMO

A fragmentação de dispositivos e sistemas operacionais móveis, aliada à grande demanda por aplicativos tem gerado a necessidade de desenvolvimento para diversas plataformas. Para execução deste processo, o mercado tem buscado ferramentas de desenvolvimento que permitam a implementação multiplataforma, garantindo maior produtividade e menor esforço. Este trabalho objetiva avaliar e comparar algumas ferramentas de desenvolvimento multiplataforma, através da criação de um aplicativo móvel, verificando alguns quesitos, permitindo identificar de forma mais evidente as principais diferenças existentes, bem como os pontos favoráveis e desfavoráveis na escolha de uma ou outra tecnologia, considerando o tempo e esforço para definição e uso de recursos dos dispositivos e sistemas operacionais móveis.



**Palavras chave:** comparativo, ferramentas, desenvolvimento, móvel, multiplataforma

## ABSTRACT

Fragmentation of mobile devices and operating systems, coupled with the high demand for applications has created the need to develop for multiple platforms. For the implementation of this process, the market has sought to develop tools enabling the multi-platform implementation, ensuring greater productivity and less effort. This study aims to evaluate and compare some cross-platform development tools, by creating a mobile application, checking some questions, allowing to identify more clearly the main differences as well as the favorable and unfavorable points in the choice of one or other technology, considering the time and effort to define and resource usage of mobile devices and operating systems.



**Key words:** comparative, tools, mobile, development, cross-platform

# LISTA DE ILUSTRAÇÕES

Figura 1: Comparativo dos diferentes tipos de aplicativos ..... 19
Figura 2: Linguagem C# como base para aplicações nativas nas plataformas móveis ..... 22
Figura 3: Desenvolvendo para diversas plataformas a partir da linguagem ..... 25
Figura 4: Criação de projeto multiplataforma com Ionic ou Cordova no Visual Studio ..... 27
Figura 5: Aplicativo ComparaPreco rodando na plataforma Android ..... 40
Figura 6: Aplicativo ComparaPreco rodando na plataforma Windows Phone ..... 41
Figura 7: Código da linguagem XAML com uso da tag OnPlatform ..... 42
Figura 8: Código da linguagem C# específico para plataforma Android ..... 43
Figura 9: Código compartilhado da linguagem C# para acesso ao recurso Câmera ..... 44
Figura 10: Opções do Visual Studio para gerar os instaladores do aplicativo ..... 46
Figura 11: Comandos visuais para execução de deploy ..... 46
Figura 12: Aplicativo rodando nos simuladores de Android, Windows e iOS ..... 49
Figura 13: Código de interface multiplataforma com a linguagem Java ..... 50
Figura 14: Configuração da simulação na IDE Eclipse ..... 53
Figura 15: Configuração de deploy na IDE Eclipse ..... 54
Figura 16: Aplicativo ComparaPreco rodando na plataforma Windows Phone ..... 57
Figura 17: Aplicativo ComparaPreco rodando na plataforma Android ..... 58
Figura 18: Aplicativo ComparaPreco rodando na plataforma iOS ..... 59
Figura 19: Definição da interface com código HTML ..... 60
Figura 20: Comparativo de tempos de instalação e tamanho dos arquivos ..... 64
Figura 21: Comparativo de tempos de desenvolvimento ..... 67

# LISTA DE TABELAS

Tabela 1: Comparativo dos trabalhos relacionados ..... 15
Tabela 2: Principais diferenças entre as ferramentas ..... 29
Tabela 3: Comparativo do processo de instalação das ferramentas ..... 63
Tabela 4: Principais características das ferramentas ..... 64
Tabela 5: Modelos de desenvolvimento ..... 65
Tabela 6: Código-fonte: Aprendizado, tempos e pontos de atenção ..... 66
Tabela 7: Comparativo de recursos para desenvolvimento de interface ..... 68
Tabela 8: Implementação de funcionalidades específicas ..... 69
Tabela 9: Testes, Deploy e distribuição ..... 69
Tabela 10: Comparativo geral ..... 70

**SUMÁRIO**

1 INTRODUÇÃO ..... 10
2 TRABALHOS RELACIONADOS ..... 13
3 FUNDAMENTAÇÃO TEÓRICA ..... 16
3.1 Sistemas Operacionais *Mobiles* ..... 16
3.2 Aplicativos móveis ..... 18
3.2.1 Desenvolvimento para Android ..... 20
3.2.2 Desenvolvimento para iOS ..... 20
3.2.3 Desenvolvimento para Windows Phone / Windows Mobile ..... 20
3.3 Desenvolvimento de aplicativos móveis com ferramentas multiplataforma ..... 21
3.3.1 Linguagem de Programação C# ..... 21
3.3.2 Xamarin ..... 22
3.3.3 Linguagem de programação Java ..... 24
3.3.4 TotalCross ..... 25
3.3.5 Linguagens de programação: HTML5, JavaScript, CSS ..... 26
3.3.6 Cordova, PhoneGap e Ionic ..... 27
3.3.4 Principais diferenças entre as ferramentas ..... 29
4 METODOLOGIA ..... 30
4.1 Obtenção, instalação e configuração ..... 30
4.2 Características ..... 31
4.3 Desenvolvimento de um aplicativo móvel ..... 31
4.4 Desenho de interface ..... 32
4.5 Testes do Aplicativo ..... 33
4.6 Distribuição ..... 34
4.7 Desempenho do aplicativo móvel ..... 34
4.8 Avaliação Geral ..... 35
5 TRABALHO DESENVOLVIDO ..... 36
5.1 Ferramentas e versões utilizadas ..... 36
5.2 Equipamentos utilizados ..... 37
5.3 Ferramentas de Desenvolvimento Avaliadas ..... 37
5.3.1 Xamarin ..... 38
5.3.1.1 Aprendizado ..... 42
5.3.1.2 Desenvolvimento de funcionalidades específicas ..... 42

5.3.1.3 Testes com o aplicativo ..... 45
5.3.1.4 *Deploy* e distribuição ..... 45
5.3.2 TotalCross ..... 47
5.3.2.1 Aprendizado ..... 50
5.3.2.2 Desenvolvimento de funcionalidades específicas ..... 51
5.3.2.3 Testes com o aplicativo ..... 52
5.3.2.4 *Deploy* e Distribuição ..... 53
5.3.3 Cordova, PhoneGap e Ionic ..... 54
5.3.3.1 Aprendizado ..... 60
5.3.3.2 Desenvolvimento de funcionalidades específicas ..... 60
5.3.3.3 Testes com o aplicativo ..... 62
5.3.3.4 *Deploy* e distribuição ..... 62
6 RESULTADOS ..... 63
6.1 Instalação e configuração ..... 63
6.2 Características ..... 64
6.2.1 Modelos de desenvolvimento multiplataforma ..... 65
6.3 Código-fonte, aprendizado e tempos de desenvolvimento ..... 65
6.4 Desenvolvimento da Interface ..... 67
6.5 Desenvolvimento de funcionalidades específicas ..... 68
6.6 Testes de aplicativos, *deploy* e distribuição ..... 69
6.7 Avaliação Geral ..... 69
7 CONCLUSÃO E TRABALHOS FUTUROS ..... 73
8 REFERÊNCIAS ..... 75

# 1 INTRODUÇÃO

A utilização de dispositivos móveis, principalmente dos telefones inteligentes (*smartphones*) tem crescido consideravelmente nos últimos anos. Isso fica evidente quando se observa as pessoas nas ruas e estabelecimentos, a maioria concentrada no uso de telefones celulares. As estatísticas oficiais confirmam que em junho de 2015 já havia 7,1 bilhões de linhas móveis no mundo todo e a projeção era de continuar a aumentar, ultrapassando inclusive o número total de habitantes do planeta (SHIRKY, 2012).

Com o aumento do uso de *smartphones*, a tendência natural dos usuários é aumentar a conectividade com a Internet. Esse fato faz com que as operadoras de telefonia móvel tenham maior demanda por pacotes de dados, permitindo aos usuários ficarem conectados através de dados móveis, dependendo menos de conexões *Wi-fi*. Cerca de 76% dos usuários que acessam a Internet no Brasil o fazem pelo *smartphone*, além disso, 63% do tempo *on-line* é feito em dispositivos móveis (LEVY, 2015).

Além do maior consumo de dados móveis, os usuários têm utilizado muito os aplicativos de celular e dispositivos móveis, gerando grande procura, não apenas dos usuários, mas de empresas e pessoas que querem disponibilizar as ferramentas para que seus serviços possam ser ofertados ou até fornecidos através de um aplicativo nos dispositivos móveis. Em 2015 o aumento no uso de aplicativos móveis foi de 58% em relação a 2014, além disso, 40% do uso foi realizado por usuários antigos (MOBILETIME, 2016).

Atualmente não há um único sistema operacional para dispositivos móveis, dependendo do fabricante é adotado um ou outro sistema. Segundo o IDC (2015), o mercado mundial divide-se da seguinte forma em relação aos três sistemas operacionais móveis mais significativos: Android, com 81,2% do mercado, iOS, com 15,8% do mercado e Windows Phone, com 2,2%. Além disso, conforme estimativa do próprio instituto, até 2019 o cenário deve permanecer muito parecido com o atual.

Desta forma, antes de iniciar o desenvolvimento de um aplicativo para dispositivos móveis, as empresas e desenvolvedores precisam preocupar-se em gerar uma aplicação que possa atender, pelo menos, os sistemas com maior relevância de mercado. O objetivo maior é conseguir fazê-lo com o menor esforço possível, bem como que o processo de geração da aplicação para os diferentes sistemas possa ser realizado de maneira transparente para os desenvolvedores e usuários.

Com esta demanda, têm surgido no mercado ferramentas que permitem aos desenvolvedores criar uma aplicação e portá-la rapidamente para os diferentes sistemas operacionais móveis existentes. Estas ferramentas são bastante recentes, das quais pode-se destacar Xamarin, TotalCross, Cordova, PhoneGap e Ionic.

A necessidade de maior agilidade no desenvolvimento dos aplicativos, bem como a rápida disponibilização nas diferentes plataformas móveis faz com que, cada vez mais, ferramentas de desenvolvimento multiplataforma sejam opções que permitam aumentar a competitividade e suprir o mercado com os aplicativos que as empresas e usuários precisam. Com o atual cenário econômico e a competição existente entre empresas do setor de tecnologia e desenvolvimento de *software* para o mercado *mobile*, a entrega de produtos e serviços no prazo, com a qualidade esperada pelo cliente tornam-se diferenciais importantes, garantindo destaque perante os concorrentes. Porém, ainda mais impactante, é a disponibilização de aplicativos que possam ser distribuídos e executados nos três principais sistemas operacionais móveis do mercado.

Assim, um dos fatores que justificam este trabalho é a possiblidade de melhorar o desenvolvimento de aplicativos para dispositivos móveis através da adoção de uma ferramenta que permita a escrita de um único código-fonte que será o gerador de aplicações para diferentes sistemas operacionais móveis. Isto permitiria reduzir os esforços e custos de desenvolvimento, tanto de novas aplicações como da conversão de aplicativos já existentes para o conceito multiplataforma, tornando as empresas e desenvolvedores mais competitivos no mercado.

Inegavelmente, a sociedade é beneficiada, à medida que a grande maioria dos usuários de dispositivos móveis tem à disposição os aplicativos, produzidos de forma democrática, para os diferentes sistemas operacionais existentes, atingindo os diferentes fabricantes e marcas de aparelhos. A inclusão social atinge um novo patamar, no qual não é preciso adquirir um aparelho moderno e inacessível à maioria da população, para ter acesso a determinado aplicativo.

Como justificativa científica pode-se afirmar que serão aprofundados os conhecimentos sobre as principais ferramentas de desenvolvimento móvel multiplataforma, não tanto especificamente ao seu funcionamento e uso, mas sim, a promessa de que, realmente, há uma enorme redução na complexidade do desenvolvimento para diferentes plataformas móveis, assim como, pode ser escrito um único código-fonte e desenhada uma única interface de usuário. Paralelamente, será indicada a ferramenta ou as ferramentas que demandam menor esforço e oferecem melhor resultado para o desenvolvedor.

Apesar de ter localizado diversos trabalhos, nos quais foram feitos comparativos de ferramentas de desenvolvimento *mobile* multiplataforma, a maior parte deles tem como foco os *frameworks* que utilizam a linguagem HTML5 em conjunto com CSS e JavaScript, possivelmente pela familiaridade das linguagens empregadas nestas plataformas, além de terem sido as primeiras a possibilitarem este tipo de desenvolvimento. Neste sentido, este trabalho tem como inovação a possibilidade de comparar estas diferentes vertentes de desenvolvimento *mobile* multiplataforma, desde o desenvolvimento com as linguagens HTML, CSS e JavaScript, que facilita, principalmente, a implementação da interface, o desenvolvimento para executar em uma máquina virtual, no qual a execução em multiplataforma praticamente resume-se ao dinamismo da máquina virtual e, por fim, a codificação de interface e lógica em determinada linguagem que faz uso das API's nativas de cada plataforma dependendo da necessidade.

Assim, pretende-se responder: qual (ou quais) dentre as ferramentas avaliadas, representa a melhor escolha, permitindo uma especificação, uma interface, um código-fonte, gerando uma boa aplicação para cada uma das três plataformas móveis mais expressivas do mercado atual?

O trabalho está organizado na seguinte estrutura: o Capítulo 2 apresenta os trabalhos relacionados pesquisados, que serviram de base para a realização deste trabalho, além de permitir identificar os diferenciais deste perante os mesmos, o Capítulo 3 contém um breve histórico dos sistemas móveis e o referencial relativo ao desenvolvimento de aplicativos com as ferramentas nativas de cada plataforma, bem como das ferramentas de desenvolvimento multiplataforma que serão comparadas e analisadas; o Capítulo 4 descreve a metodologia utilizado no trabalho, considerando o comparativo e análise das ferramentas que será desenvolvido; o Capítulo 5 contém o desenvolvimento do trabalho realizado e, o Capítulo 6 apresenta os resultados obtidos durante o desenvolvimento. Por fim, é apresentada a Conclusão e as sugestões de trabalhos futuros.

# 2 TRABALHOS RELACIONADOS

Foram pesquisados trabalhos relacionados ao tema proposto, especificamente a testes e análises comparativas de ferramentas de desenvolvimento multiplataforma visando criar uma tabela com os pontos relevantes de cada um. Observou-se que o foco dos trabalhos é limitado, tanto em relação aos sistemas móveis, como nas ferramentas testadas, que se restringiram ao desenvolvimento de aplicativos *web app* ou híbridos, com geração de *web views* ou *web conteiners*.

O trabalho realizado por Fonseca & Beder (2015) é focado na comparação entre o modelo de desenvolvimento nativo e o modelo de desenvolvimento em tecnologias web. A análise feita considera apenas a plataforma Android. Neste sentido o trabalho é bastante limitado, tanto em relação às ferramentas avaliadas, como nos sistemas operacionais, para os quais, é feita a implementação do aplicativo. Seus resultados não apontam qual é o melhor modelo a ser utilizado no desenvolvimento, ainda que o escopo seja bem limitado e específico.

A comparação efetuada por Souza (2014) é mais abrangente, já que considera os sistemas operacionais Android, iOS e Windows Phone como destinos do aplicativo desenvolvido. Apesar disso, as ferramentas analisadas utilizam o modelo de desenvolvimento *web app*, fazendo com que a análise fique restrita ao mesmo. A conclusão do trabalho revela que uma das ferramentas apresentou melhor resultado que as demais.

Café (2012) realizou um trabalho diferente, no qual recriou um aplicativo nativo utilizando a ferramenta Titanium, no modelo *web app*. As plataformas destino do aplicativo desenvolvido restringiram-se ao Android e iOS. A conclusão do estudo foi de que a ferramenta Titanium precisa melhorar, principalmente em relação a utilização dos recursos nativos dos dispositivos móveis.

Prezotto e Boniati (2014) fizeram um estudo e análise de *frameworks* de desenvolvimento para desenvolvimento de aplicativos híbridos, especificamente *web apps*. Na verdade, o estudo concentrou-se na utilização da ferramenta Cordova porém, analisando aspectos gerais no modelo de desenvolvimento multiplataforma de *web apps*. O desenvolvimento do aplicativo teve como plataformas destino: o Firefox OS, Windows Phone, Android e Blackberry. Os resultados apresentados apontaram que o modelo de desenvolvimento da ferramenta Cordova possui todos os recursos necessários, permitindo que substituía o formato de desenvolvimento nativo.

Todos os trabalhos limitaram-se, principalmente, ao formato de desenvolvimento *web app*. Além disso, apenas um dos trabalhos executou um comparativo entre diversas ferramentas, enquanto dois trabalhos apresentaram um estudo de caso, efetuando o desenvolvimento com uma única ferramenta e, um trabalho efetuou uma comparação entre o desenvolvimento *web app* e o desenvolvimento nativo.

A proposta deste trabalho é realizar uma análise comparativa, não apenas de várias ferramentas, mas também, de diferentes modelos de desenvolvimento multiplataforma. Serão avaliados os modelos de desenvolvimento: híbrido (quase nativo), proporcionado pela ferramenta Xamarin; encapsulamento do aplicativo em uma máquina virtual, representado pela ferramenta TotalCross; e *web app*, representado pelas ferramentas Cordova, PhoneGap e Ionic.

Assim, o objetivo principal deste trabalho é avaliar o processo de desenvolvimento multiplataforma com maior abrangência, não restrita às ferramentas e aos modelos de desenvolvimento, mas, estendendo-se também às plataformas móveis onde o aplicativo móvel pode ser executado. Os sistemas operacionais destinos do aplicativo serão o Android, iOS e Windows Phone, garantindo um vínculo mais condizente com a realidade atual, na qual, estas plataformas possuem a maior relevância de usuários. Para a execução da análise será desenvolvido um aplicativo móvel, cujo propósito é comparação de preços.

Os objetivos específicos a serem atingidos são:

- Avaliar a facilidade de instalação e configuração das ferramentas de desenvolvimento multiplataforma;
- Indicar como funciona o acesso aos recursos nativos em cada plataforma e quando há exceções para tratar;
- Quantificar o tempo de desenvolvimento necessário para implementar recursos específicos em cada plataforma;
- Validar a criação de uma única interface de aplicação respeitando a usabilidade e as particularidades de cada um dos três sistemas móveis, identificando exceções que precisam ser tratadas pelo desenvolvedor por não serem resolvidas pela ferramenta;
- Avaliar o desenvolvimento de recursos específicos tais como: consumo de *web-services*, recurso “compartilhar” e persistência de dados em banco de dados local, identificando necessidades ou tratamentos específicos para as três plataformas;
- Validar a necessidade de incorporação de bibliotecas específicas no processo de

geração do aplicativo para disponibilização e funcionamento em cada plataforma;

• Verificar o desempenho da aplicação móvel em operações específicas, avaliando se difere dependendo da ferramenta de desenvolvimento utilizada;

• Comparar de maneira geral, utilizando os critérios acima, as ferramentas identificando as diferenças, vantagens e desvantagens de cada uma.

A Tabela 1 apresenta um comparativo dos trabalhos relacionados que foram estudados, destacando também as diferenças em relação ao atual estudo.

**Tabela 1: Comparativo dos trabalhos relacionados**

| Autores | Objetivo | Abrangência | Tipo de comparação | Resultados |
|---|---|---|---|---|
| FONSECA, Marcos Roberto da; BEDER, Delano Medeiros. (2015) | Análise sobre qual o melhor modelo de desenvolvimento para Android, o nativo ou baseado em tecnologias *web*. | Sistema Android<br><br>Desenvolvimento Nativo<br>Desenvolvimento *Web App* | Desenvolvimento Nativo versus Desenvolvimento *Web App* | Segundo os autores não há um modelo melhor ou pior, o resultado final dependerá muito mais de um bom projeto para criar um aplicativo de qualidade. |
| SOUZA, Édipo da Silva. (2016) | Analisar as diferenças e limitações de três *frameworks* de desenvolvimento multiplataforma: Phonegap, Sencha Touch e Titanium | Sistemas Android, iOS e Windows Phone<br><br>Desenvolvimento *Web App* | Comparação de Ferramentas de desenvolvimento multiplataforma *Web App* | A conclusão apontou que uma das ferramentas se destacou das demais, não apenas pelo resultado obtido no desenvolvimento, mas também em virtude de suas características. |
| CAFÉ, Adriel Almeida. (2012) | Testar o desenvolvimento multiplataforma *web app* através de um estudo de caso refazendo um aplicativo nativo | Sistemas Android e iOS<br><br>Desenvolvimento multiplataforma *Web App* com Ferramenta Titanium | Estudo de caso recriando um aplicativo nativo | O resultado indicou que as ferramentas para criação de aplicativos *web app* ainda precisam melhorar, principalmente no acesso a recursos do dispositivo. |
| PREZOTTO, Ezequiel Douglas, BONIATI, Bruno Batista. (2014) | Estudar e analisar os *frameworks* para desenvolvimento de aplicações híbridas, bem como características e desempenho | Sistemas Firefox OS, Windows Phone, Android e Blackberry<br><br>Desenvolvimento multiplataforma *Web App* com ferramenta Cordova | Estudo de caso criando um aplicativo | A conclusão é de que as ferramentas de desenvolvimento multiplataforma *web app* podem ser utilizadas e dispõem de todos os recursos necessários para a criação de um aplicativo. |
| Este trabalho | Analisar ferramentas de desenvolvimento multiplataforma com diferentes modelos de desenvolvimento | Sistemas Android, iOS e Windows Phone Desenvolvimento (com diferentes modelos) multiplataforma com Xamarin, TotalCross, Cordova, PhoneGap e Ionic | Comparação de Ferramentas de desenvolvimento multiplataforma | Identificar a(s) melhor(es) ferramenta(s) de desenvolvimento multiplataforma que possa(m) substituir as ferramentas nativas |

Fonte: Elaborada pelo autor.

# 3 FUNDAMENTAÇÃO TEÓRICA

Os *smartphones*, altamente populares e difundidos na atualidade, nem sempre tiveram a aparência, características e usabilidades atuais. Desde o primeiro telefone celular, o Motorola Dynatac 8000X, lançado por Martin Cooper, diretor da Motorola, em 03 de abril de 1974, pesado, grande e com pouca autonomia, que levou dez anos para ser vendido comercialmente, houve muita evolução, passando pelos diferentes formatos, com teclados completos, telas sensíveis ao toque e, por fim, a, praticamente, ausência de botões físicos (BRIGGS, 2016).

As mudanças não ficaram restritas à aparência, muito além disso, as novas tecnologias e miniaturização dos circuitos permitiram a agregação de diferentes recursos ao telefone, começando pelas agendas eletrônicas, agenda de compromissos, visualizadores de documentos, clientes de e-mail, navegadores de internet, câmeras digitais integradas, tocadores de música, serviço de GPS (do inglês *Global Positioning System*, traduzido em Sistema de Posicionamento Global), modem de internet, comunicadores instantâneos, clientes VPN (do inglês *Virtual Private Network*, traduzido em Rede Virtual Privada), suporte a cartões de memória externos (MORIMOTO, 2009).

Para a maioria das pessoas pode ser complicado diferenciar um simples telefone de um *smartphone*, comumente chamado de "telefone inteligente", pois a diferença às vezes é tênue ou pouco evidente. Segundo Morimoto (2009), um telefone com *status* de *smartphone* deve ser capaz de: executar um sistema operacional completo, permitindo instalação de aplicativos nativos, comunicar-se com um PC via USB/Bluetooth, acessar a internet e possuir ferramentas para tal, além de tocar músicas, exibir vídeos e executar jogos.

## 3.1 Sistemas Operacionais *Mobiles*

Um dos primeiros sistemas operacionais utilizados em telefones e *smartphones* com grande sucesso mundial foi o Symbian OS, surgido no final dos anos 90 e início dos anos 2000 e originário do EPOC, sistema cujo diferencial era a excelente utilização dos poucos recursos de *hardware* disponíveis na época mantendo um ótimo desempenho. O Symbian OS foi adotado pelos principais fabricantes de telefones como Nokia, Motorola, LG, Samsung, Sony-Ericsson e permitia o desenvolvimento de aplicativos utilizando as linguagens C e Java. Foi muito fomentado pela Nokia, que inclusive adquiriu a empresa Symbian e a transformou em Symbian Fundation, visando neutralidade, bem como abrindo o código fonte do sistema (ZUCKER, 2011).

Na mesma época, em torno de 2002, o Windows Pocket PC 2002 era apresentado pela Microsoft, já com suporte à telefonia e visual que lembrava o Windows XP utilizado em computadores *desktops*. Alguns equipamentos que executavam o Windows Pocket PC podiam ser operados com o auxílio de uma caneta, possuindo tela sensível ao toque. Até o final da primeira década do século XXI, sucederam-se versões, com pequenas mudanças na interface e de nome, sendo a última versão com este formato o Windows Mobile 6.5, lançado em 2009 (THOMAZ, 2015).

A partir de 2010, com o lançamento do Windows Phone 7, totalmente remodelado, a Microsoft busca focar no conceito *touch* das telas dos dispositivos, com o objetivo de tornar a experiência do usuário mais agradável e fácil. Em 2013, atualiza novamente seu sistema, lançando o Windows Phone 8 e 8.1, novamente reescrito e com novos recursos, com o objetivo de se manter no mercado *mobile* (MÔNACO, 2014).

Em 2015, inicia o desenvolvimento do Windows 10 Mobile, com um formato inovador, baseado no *feedback* de usuários, além de criar um sistema único (*mobile* e *desktop*). Desde as primeiras versões do Windows Pocket, passando pelo Windows Phone e, até o Windows 10 Mobile, os aplicativos para o sistema podem ser escritos usando a plataforma .NET, da própria Microsoft.

No ano de 2003, estabelecia-se na Califórnia a empresa Android Inc. Seu objetivo inicial era criar um sistema para câmeras digitais, porém, por limitações deste mercado, acabaram mudando o foco para o *mobile*. A empresa Google apostou na ideia e, em 2005, adquiriu a Android Inc., para, em 2008, em parceria com a HTC, lançar comercialmente o primeiro telefone equipado com Android. Na época foi desacreditado pelos líderes de mercado como Nokia, Microsoft e Apple, porém, sendo de código-aberto e altamente fomentado pela Google, tanto na parceria com fabricantes de telefones, como no incentivo aos desenvolvedores de aplicativos, rapidamente dominou o mercado (MEYER, 2015). Atualmente, o Android está na versão 7, denominada também como “Nougat”, sendo que, para o desenvolvimento de aplicativos, é utilizada a linguagem Java.

Em 9 de janeiro de 2007 foi apresentado ao mundo pela Apple, o iPhone 1 e, junto com ele, o iOS, sistema utilizado no equipamento. O iOS possuía aplicações bastante inovadoras, por estarem em um telefone e, aliado a um equipamento de qualidade, rapidamente caiu nas graças dos consumidores, chegando ao ponto de muitos considerarem um divisor de águas no segmento, mudando o conceito de *smartphone* (LECHETA, 2016).

O desenvolvimento de aplicativos para o iOS é realizado com a linguagem Objective-C e, recentemente, também pode ser feito com a linguagem Swift, apresentada e adotada pela Apple como a nova linguagem de programação para o iOS.

Destes sistemas operacionais, os com desenvolvimento ativo e relevância de mercado são o Android, o iOS e o Windows Phone.

### 3.2 Aplicativos móveis

Segundo Souza (2014) um aplicativo móvel é "...um software destinado a ser executado em um dispositivo móvel, como em um smartphone, e é obtido normalmente através das lojas de aplicativos existentes em cada plataforma". Evidentemente, além deste conceito básico, o aplicativo tem por objetivo suprir alguma necessidade dos usuários, desde a diversão, com um aplicativo-jogo, trabalho, com um aplicativo de cadastro de pedidos até ferramenta, com um aplicativo para economizar bateria.

Diferente do que ocorre atualmente, no final dos anos 90 e início dos anos 2000, quando o sistema operacional móvel Symbian dominava o mercado, o desenvolvimento e distribuição de aplicativos móveis era realizado de uma forma diferente. O próprio desenvolvimento era bastante limitado, principalmente pelo fato de os próprios fabricantes produzirem os aplicativos que seriam embarcados em seus equipamentos. Com o aumento na demanda de novos e cada vez mais complexos aplicativos e jogos, foram desenvolvidos novos recursos e *frameworks*, que permitiram não apenas facilitar o processo, mas também, engajar desenvolvedores independentes a criarem aplicativos.

A distribuição dos aplicativos também era bastante limitada, pois dava-se em geral, de maneiras diversas e pouco centralizadas, salvo quando o aplicativo era embarcado no equipamento pelo fabricante. Costumeiramente, as aplicações eram instaladas pela operadora de telefonia antes da distribuição e venda dos telefones ou ainda, posteriormente, sob demanda dos usuários, através de troca de mensagens ou lojas rudimentares da operadora. Em situações específicas e menos comuns o desenvolvedor, manualmente, efetuava a instalação diretamente em dispositivos objetivando quase sempre, o atendimento a um cliente específico que seria o usuário do aplicativo desenvolvido.

Com a chegada do iPhone, em 2007, e a criação da loja Apple Store de aplicativos no ano seguinte, que também representou um sucesso instantâneo, os demais fabricantes logo identificaram uma necessidade de atender a nova demanda. Rapidamente o Google lançou o

Android Market e, em 2009, a Nokia lançou a Ovi Store para concorrer com a loja Apple Store (ZUCKER, 2011). Na mesma tendência a Microsoft anuncia o Windows Phone 7 no ano de 2010, já com a Microsoft Marketplace.

As lojas de aplicativos facilitaram o encontro do desenvolvedor com os seus clientes usuários e suas necessidades. O sucesso e o grande retorno financeiro proporcionado por alguns aplicativos rapidamente fizeram com que mais desenvolvedores fossem atraídos para o segmento, além disso, o próprio Google foi grande fomentador do desenvolvimento de aplicativos móveis, principalmente para apoiar seu próprio sistema operacional *mobile*, o Android. Hoje, os desenvolvedores podem publicar seus aplicativos nas três lojas. Dessa forma, atingem imediatamente todos os usuários dos três sistemas operacionais *mobiles* mais significativos do mercado.

Segundo Budiu (2013) "os aplicativos *mobiles* podem ser de três tipos: nativos, híbridos ou *web apps"*. Os aplicativos nativos são instalados no dispositivo e conseguem usufruir de todos os recursos e funcionalidades do sistema operacional e do dispositivo, já o aplicativo *web app* é um *site* exibido em um *web contêiner* ou no próprio navegador do dispositivo que, pode acessar poucos recursos do dispositivo e sistema operacional e, por fim, o aplicativo híbrido é um intermediário entre o aplicativo nativo e o *web app*, pois, em geral, executa em um *web contêiner* e possui acesso a diversos recursos do sistema operacional e dispositivo móvel.

A Figura 1 faz um comparativo que indica de maneira mais clara as principais diferenças entre os três tipos de aplicativo. São comparados 5 quesitos importantes no desenvolvimento, que são: acesso aos recursos do dispositivo, performance do aplicativo, tempo de desenvolvimento, disponibilizado na loja de aplicativos (*app store*), executa em diversas plataformas (multiplataforma).

**Figura 1: Comparativo dos diferentes tipos de aplicativos**

| | Acesso ao Dispositivo | Performance | Tempo Desenvolvimento | App Store | Multi Plataforma |
|---|---|---|---|---|---|
| **Nativo** | **Sim** | **Sim** | **Caro** | **Sim** | **Não** |
| **Web** | **Parcial** | **Sim** | **Ótimo** | **Não** | **Sim** |
| **Híbrido** | **Sim** | **Sim** | **Ótimo** | **Sim** | **Sim** |

Fonte: Prezotto & Boniati, 2014.

### 3.2.1 Desenvolvimento para Android

Como opção de desenvolvimento nativa para Android, o Google disponibiliza a ferramenta Android Studio, na qual o desenvolvedor utiliza a linguagem de programação Java na escrita do código-fonte, e a linguagem XML para definição do *layout* de interface do aplicativo.

Segundo o Google (2016), o Android Studio "é um Ambiente de desenvolvimento integrado oficial para o desenvolvimento de aplicativos Android e integra todos os recursos necessários e possíveis para garantir a produtividade na criação das aplicações". Ele possui suporte e pode ser executado nos sistemas operacionais *desktop* Windows, Linux e Mac OS X.

### 3.2.2 Desenvolvimento para iOS

O ambiente de desenvolvimento integrado nativo para iOS é o Xcode, a linguagem de programação para a criação de aplicativos é a Swift e o próprio ambiente possui uma ferramenta para fazer a criação e desenho da interface da aplicação.

Conforme a Apple (2016) "o Xcode fornece aos desenvolvedores um fluxo de trabalho unificado para design de interface do usuário, codificação, testes e depuração" sendo um ambiente de desenvolvimento totalmente integrado com as tecnologias empregadas em seus dispositivos, garantindo produtividade e facilidade na criação de aplicações.

### 3.2.3 Desenvolvimento para Windows Phone / Windows Mobile

A ferramenta nativa para desenvolvimento de aplicativos para o Windows Phone / Windows Mobile é o ambiente de desenvolvimento integrado Visual Studio 2015. A implementação das aplicações *mobiles* pode ser feita utilizando a linguagem C# ou a tecnologia Silverlight, já o desenho da interface pode ser realizado com o recurso "*drag and drop*" (arrastar e soltar) ou com linguagem XML/XAML.

De acordo com a Microsoft (2016), o Visual Studio 2015 "é um ambiente completo que tem todos os recursos necessários, permitindo ao desenvolvedor codificar, depurar e testar seus aplicativos móveis". Ele também permite criar aplicativos universais, que executam nos celulares, *desktops*, *tablets* e no Xbox One, cujo sistema operacional seja o Windows 10. Apesar da versão tradicional ser suportada apenas em Windows, há uma versão específica, denominada Visual Studio Code, com suporte para ser instalada em Windows, Mac OS ou Linux.

## 3.3 Desenvolvimento de aplicativos móveis com ferramentas multiplataforma

Conforme Petzold (2015), são quatro os principais problemas no desenvolvimento de aplicativos móveis para as três principais plataformas do mercado. O primeiro é a diferença de interface com usuário em cada um dos três sistemas, seja pela presença de diferentes botões físicos e capacitivos ou posicionamento de menus e opções para o usuário. O segundo é que cada sistema possui uma IDE (do inglês *Integrated Development Environment*, traduzido em Ambiente de Desenvolvimento Integrado). O terceiro é a diferente interface de programação, já que cada sistema possui uma determinada API (do inglês *Application Programming Interface*, traduzido em Interface de Programação de Aplicativos), fazendo com que cada plataforma possa dar nomes diferentes para um mesmo tipo de objeto de interface do usuário. O quarto é que cada sistema possui sua própria linguagem de programação para desenvolver as aplicações, no iOS é a Objective-C, no Android é o Java e no Windows Phone é a plataforma .NET, na qual podem ser utilizadas as linguagens C#, Visual Basic ou C++.

Devido a fragmentação de dispositivos e sistemas operacionais *mobiles*, os desenvolvedores precisam construir a mesma aplicação móvel para cada um deles, fazendo com que o processo de implementação e liberação da mesma demore mais que o esperado ou, então, privilegie usuários de determinado sistema. Assim, a necessidade de conseguir gerar um único aplicativo móvel que possa ser executado nos diferentes dispositivos e sistemas *mobile* é cada vez maior, buscando-se alternativas para que isto seja possível (PETZOLD, 2015).

Esta necessidade impulsiona não apenas os desenvolvedores, mas empresas e grupos a buscarem soluções que permitam o desenvolvimento unificado para diversas plataformas móveis, garantindo que o tempo de criação e distribuição do aplicativo seja adequado e, ao mesmo tempo atenda aos requisitos de qualidade, usabilidade e desempenho esperados.

### 3.3.1 Linguagem de Programação C#

A linguagem C# é relativamente nova, quando comparada ao Objective-C e Java, tendo sido apresentada no ano 2000 pela Microsoft, fortemente tipada, orientada a objetos e simples, influenciada pela linguagem C++ e Java, porém mais limpa e sem bagagem histórica. Sua íntima ligação com o .NET desde a criação, além de fornecer infraestrutura para os tipos básicos, permite a utilização da extensa biblioteca de classes, suportando diferentes tipos de programação, desde escrita e leitura de arquivos a programação concorrente (PETZOLD, 2015).

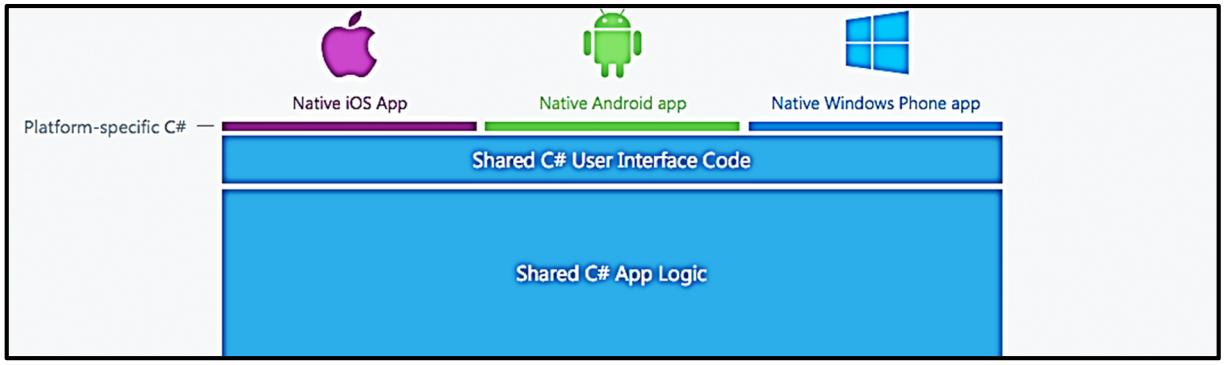
A criação da nova linguagem pela Microsoft permitiu potencializar ainda mais a

plataforma .NET, através da qual as aplicações criadas podem facilmente, executar em dispositivos diversos que não sejam, necessariamente, computadores tradicionais. Além disso, suas origens estão especificamente ligadas as linguagens C, C++ e Java, aproveitando seus melhores recursos, bem como, facilitando o aprendizado dos programadores (DEITEL *et al.*, 2003).

Conforme Silva (2015), "a popularidade da linguagem C# tem aumentado no desenvolvimento dos mais diversos tipos de software, inclusive no desenvolvimento de aplicativos mobile", pois C# é a linguagem base utilizada no desenvolvimento com Xamarin, permitindo o desenvolvimento unificado para Android, iOS e Windows Phone.

Na Figura 2 pode-se visualizar o modelo de funcionamento do Xamarin, no qual o código-fonte e o código de interface do usuário são escritos na linguagem C# e, através do .NET *framework mono,* são convertidos para código nativo em cada plataforma de maneira transparente, tanto para o desenvolvedor como para o usuário.

**Figura 2: Linguagem C# como base para aplicações nativas nas plataformas móveis**




Fonte: Microsoft, 2016.

### 3.3.2 Xamarin

A ferramenta Xamarin teve seu desenvolvimento iniciado em 2011 pela empresa de mesmo nome, que mais tarde foi adquirida pela Microsoft. Os principais recursos disponíveis na ferramenta são: a linguagem C#, bastante difundida, permitindo utilizar a sintaxe que já é familiar; .NET *framework mono*, que é a implementação de plataforma cruzada do *framework* .NET, ou seja, é o que permite ao desenvolvedor utilizar código nativo da plataforma .NET, que é convertido para código nativo de outras plataformas (este processo também é denominado como *cross plataform* ou plataforma cruzada); compilador, que produz um aplicativo nativo ou interpretado em tempo de execução, quando possível; a própria IDE e o *plugin* para o Visual Studio que serve de base para projetos Xamarin. Estes recursos reunidos permitem o

desenvolvimento de aplicativos com execução nativa nos sistemas iOS, Android e Windows Mobile, além de outros que não serão objeto neste estudo (MICROSOFT, 2016).

Em relação a compilação dos aplicativos móveis produzidos na Linguagem C# há diferenças dependendo da plataforma destino. Para iOS, o aplicativo é compilado a partir da pré-compilação do modo a*head of time* (antes do tempo) em linguagem do tipo ARM (do inglês A*dvanced RISC Machine* traduzido em Máquina Avançada *RISC,* cujo significado é *chip* com conjunto de instruções reduzido), que, devido a limitação da Apple em não permitir a geração de código em tempo de execução, não pode ter todas as características utilizadas. Em outras palavras, a compilação é definitiva e o aplicativo não poderá gerar ou processar código em tempo de execução. Para Android, o aplicativo é compilado para IL (do ingês Microsoft *Intermediate Language* traduzido em Linguagem Intermediária Microsoft), processado em tempo de execução pela monoVM (máquina virtual mono .NET) e interagindo com Java e Dalvik (máquina virtual nativa do Android que executa os aplicativos) através de JNI (do inglês *Java Native Interface* traduzido em Interface Nativa Java). De forma simplificada, o código fonte é parcialmente interpretado em tempo de execução. Já no Windows, o aplicativo também é compilado para IL e processado em tempo de execução sem o auxílio de outros recursos (MICROSOFT, 2016).

Mesmo que a linguagem de desenvolvimento seja a C#, a plataforma Xamarin provê acesso a recursos do CocaTouch SDK do iOS, através do Xamarin.iOS e do Google Android SDK, através do Xamarin.Android. Para o Windows a linguagem C# já contém todos os recursos do SDK nativamente.

A ferramenta Xamarin permite a criação de uma interface nativa e específica para cada plataforma, porém, nesse estudo o foco será a utilização do Xamarin.Forms que permite o desenho de uma única interface para todas as plataformas. Evidentemente, este recurso faz os ajustes necessários para que as telas sejam apresentadas ao usuário de cada plataforma com os elementos visuais e características específicas de cada uma.

É possível efetuar os testes dos aplicativos diretamente a partir da ferramenta Xamarin, utilizando simuladores e emuladores específicos para cada plataforma. Em relação ao iOS é necessário conectar a ferramenta com um MAC OS na rede local para que seja possível executar o emulador. Já para Android e Windows Phone, os testes podem ser feitos executando diretamente o emulador correspondente presente na ferramenta.

A plataforma Xamarin tem como diferencial as bibliotecas Xamarin.Mac, Xamarin.iOS

e Xamarin.Android que, basicamente, são versões .NET das API's nativas das plataformas iOS e Android. Desta forma, é possível desenvolver para as plataformas iOS, Android e Windows Phone (utilizando o Visual Studio), escrevendo o código em C#, acessando os recursos nativos de cada plataforma e ainda contando com todo o suporte da linguagem C# e da plataforma .NET. Além disso, conta com a biblioteca Xamarin.Forms, cujo objetivo é permitir ao desenvolvedor criar uma única interface da aplicação móvel que poderá ser compilada para qualquer uma das três plataformas, deixando a maior parte da complexidade relativa à adequação para cada plataforma móvel a cargo do Xamarin (PETZOLD, 2015).

### 3.3.3 Linguagem de programação Java

De acordo com Souza et al. (2010), "a linguagem Java foi apresentada ao mundo em 1995 e rapidamente tornou-se mundialmente conhecida, em parte, pela independência de plataforma pelo fato de executar em uma máquina virtual Java" (JVM, do inglês Java Virtual Machine traduzido em Máquina Virtual Java), bem como, da possibilidade de executar aplicações através do *browser* de internet.

Segundo Deitel (2003), "a adoção e popularização da linguagem Java ocorreu devido ao momento em que foi divulgada, justamente quando a internet despertava grande interesse, a nova linguagem, por sua vez, propiciava a criação de páginas de internet dinâmicas, permitindo o desenvolvimento de verdadeiras aplicações web para as organizações, bem como, o desenvolvimento de aplicativos para usuários executarem em dispositivos móveis".

O desenvolvimento com Java permite que um único código-fonte seja executado em diferentes plataformas, conforme ilustração da Figura 3.

**Figura 3: Desenvolvendo para diversas plataformas a partir da linguagem**


Fonte: TotalCross, 2016.

### 3.3.4 TotalCross

A ferramenta TotalCross foi lançada em janeiro de 2009 para ser o sucessor do SuperWaba, surgida em meados do ano 2000. A linguagem utilizada no desenvolvimento com TotalCross é Java, permitindo rápida familiarização, já a execução dos aplicativos ocorre na TCVM (TotalCross Virtual Machine), máquina virtual própria da ferramenta, na qual são interpretados *optcodes* proprietários, ao invés dos tradicionais Java *bytecodes*. Não há um ambiente de desenvolvimento integrado próprio, mas o TotalCross pode ser facilmente integrado ao Eclipse, que é um tradicional ambiente de desenvolvimento integrado Java, porém, também é possível fazer o desenvolvimento editando o código-fonte com um editor de texto comum. O iOS, Android e Windows são suportados pela ferramenta, além de outros que não serão estudados neste trabalho (TOTALCROSS, 2016).

É importante destacar que a grande diferença do modelo utilizado pelo TotalCross consiste na utilização da TCVM, ou seja, o aplicativo criado é um híbrido que, apesar de conseguir acesso a recursos nativos do dispositivo executa em uma máquina virtual no mesmo.

Em tempo de desenvolvimento, a execução do aplicativo pode ser feita diretamente com a máquina virtual Java que executa o TotalCross *launcher*, porém isto serve apenas para testar a aplicação. Quando for feita a compilação do aplicativo para distribuição deve ser utilizada

uma classe específica para fazer o *deploy* da aplicação, sendo que este processo gera o arquivo para a plataforma correspondente (TOTALCROSS, 2016).

Os recursos necessários ao aplicativo devem ser acionados diretamente pelas classes disponíveis para execução na TCVM, ficando a cargo da máquina virtual o acesso aos recursos do dispositivo ou sistema operacional correspondente.

A criação da interface do aplicativo é feita com os próprios recursos da linguagem Java, ou seja, o desenvolvedor, através do seu conhecimento da linguagem, utilizará as classes específicas para desenhar as telas e seus componentes. A adaptação de características para uma ou outra plataforma dependerá da destreza do programador, pois não haverá um ajuste automatizado, que dependa da plataforma para a qual o aplicativo será compilado (TOTALCROSS, 2016).

Durante o processo de desenvolvimento do aplicativo, os testes podem ser realizados continuamente de forma simples e rápida, visto que o aplicativo é executado, com o apoio do TotalCross *launcher*, diretamente pela máquina virtual Java presente na máquina do desenvolvedor, independentemente da plataforma destino. Por ter esta característica, não é necessária a presença de um emulador ou simulador específico para cada plataforma (TOTALCROSS, 2016).

Conforme TotalCross (2016), "a ferramenta permite desenvolver a aplicação, tanto em relação ao código-fonte, propriamente dito, assim como a definição da interface para o usuário, com o uso da linguagem Java, utilizando as classes e métodos disponíveis no SDK" (*Software Development Kit*, traduzido em Kit para Desenvolvimento de Software). A execução do aplicativo no dispositivo móvel é feita através de uma máquina virtual da plataforma TotalCross denominada TCVM, parecida com a JVM, porém, com melhor desempenho.

### 3.3.5 Linguagens de programação: HTML5, JavaScript, CSS

De acordo com o W3C (2014), "a linguagem HTML5 continua sendo HTML clássico, que permite marcar e descrever documentos, porém, com muitas melhorias e correções, principalmente para atender a demanda de possibilitar a criação e execução de aplicativos na internet, que até então sempre ficou a cargo de outras linguagens auxiliares".

A partir desta nova realidade e com os dispositivos móveis suportando nativamente o HTML5, mais *frameworks* e ferramentas de desenvolvimento de aplicativos têm tirado proveito da linguagem em conjunto com o CSS e Java Script. Pode-se destacar que o ponto mais

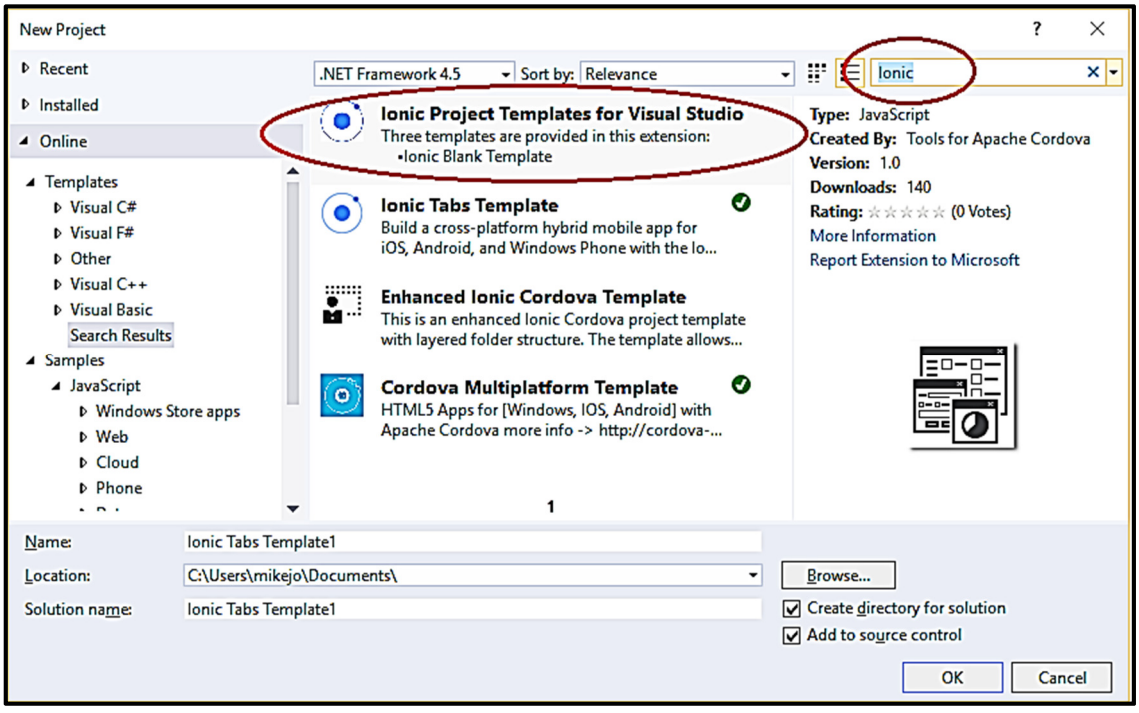
relevante é a facilidade e o poder da linguagem HTML, aliado ao CSS e JavaScript na criação da interface dos aplicativos, bem como, o conhecimento prévio destas linguagens por parte dos desenvolvedores.

### 3.3.6 Cordova, PhoneGap e Ionic

As ferramentas Cordova, PhoneGap e Ionic possuem uma estrutura semelhante, assim como, o processo de desenvolvimento de aplicativos pode ser considerado parecido. Em parte, isto deve-se ao fato das três plataformas disponibilizarem as linguagens JavaScript, HTML5 e CSS para o desenvolvimento de aplicações móveis. Contribui também o fato destas ferramentas gerarem um aplicativo híbrido ou *web app* e não um aplicativo nativo.

A Figura 4 mostra a criação de um projeto Ionic diretamente no IDE Visual Studio.

**Figura 4: Criação de projeto multiplataforma com Ionic ou Cordova no Visual Studio**




Fonte: Microsoft, 2016.

A instalação destas ferramentas tem os seguintes suportes:

Cordova: Windows, Linux e Mac OS X;

PhoneGap: Windows e Mac OS X;

Ionic: Windows, Linux e Mac OS X.

Os *frameworks* Cordova, PhoneGap e Ionic não possuem uma IDE própria e o

desenvolvimento é baseado em comandos. O PhoneGap possui uma ferramenta de apoio para facilitar o trabalho e visualização. Além disso é possível utilizar os *frameworks* integrados em um IDE, como, por exemplo, o Visual Studio (PHONEGAP, 2016).

As três ferramentas permitem que no momento da compilação seja especificada a plataforma móvel para a qual o aplicativo destina-se. Desta forma, ele é gerado na forma de um *web view* ou *web conteiner* com a aplicação embutida. Da mesma forma, o acesso a alguns recursos do sistema ou dispositivo é realizado através de *plugins* do Cordova, que são também utilizados no PhoneGap e Ionic. Dependendo do tipo de aplicativo que será desenvolvido pode ser necessário instalar os SDK's de desenvolvimento nativos das plataformas, para permitir que os *plugins* JavaScript acessem os recursos necessários (CORDOVA, 2015).

A interface é criada utilizando as linguagens HTML5 e CSS, sendo possível utilizar modelos pré-definidos já existentes para cada plataforma móvel, permitindo que a identidade visual do aplicativo corresponda ao sistema operacional onde irá executar.

Em relação aos testes e emuladores para visualizar o aplicativo móvel sendo executado, Cordova e Ionic utilizam os próprios emuladores presentes nos SDK's nativos de cada plataforma, enquanto que o PhoneGap permite a visualização do aplicativo em um *browser* diretamente no *desktop* do desenvolvedor.

Estas três ferramentas são definidas como *frameworks* para o desenvolvimento de aplicativos móveis. Conforme Souza (2014) citando Fayad e Schmidt (1997), "...um Framework é uma aplicação semi-completa, reusável que pode ser especializada para produzir aplicações customizadas". Assim, pode-se considerar que o *framework* tem como objetivos facilitar e reduzir o trabalho do desenvolvedor, por permitir a reutilização e adequação de seu código com o objetivo de atingir determinado resultado.

O *framework* Ionic é um SDK HTML5 para criação de aplicativos nativos ou híbridos para dispositivos móveis utilizando tecnologias como HTML, CSS e JavaScript. Considerando que o Ionic serve, principalmente, para descrição da parte visual do aplicativo, é possível integrá-lo facilmente com Cordova, por exemplo, para permitir a publicação do aplicativo, assim como o acesso aos recursos nativos do dispositivo móvel (IONIC, 2016).

O *framework* Apache Cordova também utiliza HTML5, CSS3 e JavaScript, porém seu diferencial reside na existência de *plugins* que possibilitam o acesso as API's nativas das plataformas móveis facilitando a utilização de recursos, tais como câmera e contatos do dispositivo móvel (CORDOVA, 2015).

Segundo PHONEGAP (2016), o Cordova *"...is still the engine that powers PhoneGap*", ou seja, o Cordova é o motor que alimenta o PhoneGap, possivelmente porque o código fonte do PhoneGap foi doado à Apache em 2011, dando origem ao Apache Cordova, que tem evoluído desde então e, consequentemente, tem compartilhado a evolução com o PhoneGap.

Efetivamente, a utilização de linguagens, relativamente simples e amplamente utilizadas, não apenas em aplicativos, mas em *sites* e páginas *web* de propósito geral ou de aplicações *web* aliado a possibilidade de utilizar *plugins* que provêm o acesso aos recursos nativos dos dispositivos, faz com que estas ferramentas sejam uma boa alternativa, garantindo produtividade e qualidade.

### 3.3.7 Principais diferenças entre as ferramentas

Analisando as definições das ferramentas apresentadas anteriormente, pode-se identificar algumas características em relação ao desenvolvimento multiplataforma: a Xamarin tem uma linguagem própria para criação de interface, denominada Xamarin.Forms, e utiliza-se de mapeamentos internos, controlados pela própria ferramenta para acessar as API's nativas das plataformas, sendo transparente para o desenvolvedor; a TotalCross utiliza a linguagem Java como modelador de interface e acessa recursos nativos através da máquina virtual própria que executa o aplicativo no dispositivo móvel; o Cordova e o PhoneGap permitem modelar a *interface* utilizando as linguagens HTML5, CSS e JavaScript, possuindo *plugins* para acessar recursos nativos do dispositivo móvel; por fim, o Ionic, que também faz uso das linguagens HTML5, CSS e JavaScript, precisa ser utilizado em conjunto com outro *framework* (Cordova, por exemplo), para que seja possível acessar recursos nativos do dispositivo móvel.

Também vale ressaltar a diferença no processo de execução dos aplicativos, pois enquanto os aplicativos gerados com Xamarin, Cordova, PhoneGap e Ionic executam diretamente nos dispositivos, os aplicativos gerados com a TotalCross executam na máquina virtual TCVM, que por sua vez comunica-se com o sistema operacional do dispositivo.

**Tabela 2: Principais diferenças entre as ferramentas**

| **Características** | **Xamarin** | **TotalCross** | **Cordova, PhoneGap, Ionic** |
|---|---|---|---|
| Criação da interface | Xamarin.Forms | Linguagem Java | Linguagens HTML5 e CSS |
| Recursos nativos | API's nativas das plataformas | TCVM – TotalCross *Virtual Machine* | *Plugins* Cordova |
| Execução | Quase nativa | Emulada na TCVM | Nativa, encapsulada em *web app* |

Fonte: Elaborada pelo autor.

# 4 METODOLOGIA

Após o estudo do desenvolvimento nativo ou multiplataforma para dispositivos e sistemas operacionais móveis este capítulo apresenta a proposta de comparação das ferramentas Xamarin, TotalCross, Cordova, PhoneGap e Ionic, avaliando os pontos positivos e negativos, bem como os diferenciais e limitações, considerando-se também o tipo de aplicativo gerado por cada ferramenta.

Segundo Heidenheimer *et al.* (1983), "uma pesquisa comparativa entre duas ou mais coisas permite a descoberta de algo". Neste trabalho, objetiva-se descobrir se umas das ferramentas é melhor que as demais, além de elencar, na forma de comparativo, as diferenças entre elas, sejam elas favoráveis ou desfavoráveis.

## 4.1 Obtenção, instalação e configuração

A obtenção, instalação e configuração da ferramenta de desenvolvimento são processos críticos, pois podem criar uma primeira barreira para o desenvolvedor. É necessário destacar que, antes de iniciar a programação, o ambiente deve ser preparado, dependendo do tamanho da organização, pelo próprio usuário da ferramenta.

A obtenção da ferramenta, com a disseminação da internet, pode ser realizada através de *download* ou, então, seguindo um modelo mais tradicional, o uso de mídias físicas fornecidas pela empresa fabricante. Assim, o tamanho do pacote de instalação torna-se um item a considerar, principalmente quando a transferência do mesmo é realizada através da internet.

Após a obtenção do pacote de instalação, é necessário executar a instalação e configuração da ferramenta, cujo processo poderá ser realizado na forma de um *wizard*, ou mesmo, através de descompactação, cópia de arquivos, edição de arquivos de configuração, dentre outros. Dependendo do método de instalação são necessários manuais ou tutoriais que apoiem a realização desta tarefa, garantindo o sucesso da atividade, bem como o uso da ferramenta.

De forma resumida, a análise e comparação deste item considerará:

- Tipo de disponibilização do pacote (mídia física / *download*);

- Tamanho do pacote;

- Processo de instalação (*wizard*, manual);

- Processo de configuração (se existe, se é automático, se é manual).

## 4.2 Características

As características das ferramentas de desenvolvimento permitem a avaliação concreta de que atendem a necessidade de um determinado projeto, visto que muitos elementos podem ser considerados, além de serem muito diversos, dependendo da organização, do cliente ou mesmo do tipo de aplicativo a ser criado. Assim, faz-se um mapeamento das características de cada ferramenta, indicando a existência ou não do recurso e, também, a descrição do recurso quando é próprio para cada ferramenta, citando como exemplo, a linguagem de programação, que apesar de sempre estar presente, poderá ser diferente.

Dentre as características mais relevantes nas ferramentas de desenvolvimento móvel pode-se citar: o licenciamento, o IDE, a linguagem de programação, desenho da interface e acesso aos recursos nativos dos sistemas operacionais e dispositivos. Com isso, serão comparadas as características a seguir:

- Licenciamento (gratuito ou pago);

- Ambiente de desenvolvimento integrado nativo;

- Linguagem de programação;

- Escrita e desenho da *interface*;

- Permite acesso aos recursos nativos das plataformas móveis.

## 4.3 Desenvolvimento de um aplicativo móvel

O desenvolvimento de aplicativo móvel consiste na codificação das instruções que deverão ser executadas pelo mesmo. Normalmente é uma das tarefas mais impactantes no tempo de execução de um projeto de desenvolvimento e, por isto, torna-se um dos itens que pode representar um enorme ganho quando realizado sob a perspectiva de multiplataforma, ou seja, um único desenvolvimento que pode ser distribuído para diversos sistemas.

Especificamente, alguns critérios permitem uma avaliação deste processo, como: curva de aprendizagem, tempo de desenvolvimento e apoio da IDE, com recursos como *auto complete*, por exemplo.

A comparação e análise do desenvolvimento abrangerá:

- Aprendizagem do uso da ferramenta (fácil, média, difícil);

- Tempo de desenvolvimento de funcionalidades específicas: cadastro com gravação em banco de dados do dispositivo, consumo de *web-service*, envio de e-mail;

- Tempo de desenvolvimento global;

- Apoio do ambiente de desenvolvimento integrado avaliando os recursos: *auto complete*, verificação de erros em tempo real, permite *debug* e *checkpoint*.

Considerando que a maior parte da comparação das ferramentas se dará, especificamente, no desenvolvimento, será implementado um aplicativo *mobile*. O trabalho de desenvolvimento deste aplicativo servirá para avaliar e comparar diversas funcionalidades das ferramentas de desenvolvimento.

O aplicativo móvel que será desenvolvido terá como função a coleta e compartilhamento colaborativo de preços, sendo identificado com o nome de "ComparaPreco". Ele terá as características básicas de um aplicativo móvel atual, considerando persistência de dados, utilização de câmera e sensores do dispositivo, consumo de *web-services* e compartilhamento de informações através de outros aplicativos.

Como requisitos fundamentais do aplicativo serão considerados os seguintes itens:

- Cadastro de estabelecimentos de venda;
- Consulta de estabelecimentos cadastrados;
- Cadastro de produtos e preços;
- Consulta de produtos e preços cadastrados;
- Consumo de um *web-service* no formato JSON;
- Captura de foto de produto com a câmera do dispositivo;
- Compartilhamento de produtos e preços utilizando aplicativos de terceiros.

## 4.4 Desenho de interface

A interface sempre esteve em destaque no desenvolvimento de aplicações e, com aplicativos móveis, a interface tem relevância ainda maior, pois devem ser considerados detalhes tais como usabilidade para determinada plataforma, recursos de *hardware* com os quais o usuário interage com o aplicativo, padrões de interface diferentes para cada sistema operacional móvel, costume do usuário com determinada interface padrão dos aplicativos.

Percebe-se que, apesar de parecer trivial, a definição da interface de um aplicativo móvel não é simples e deve ser realizada observando diversos aspectos importantes, com preocupações não apenas com o usuário, mas, também, com o sistema onde a aplicação será executada. Desta forma, as ferramentas podem fornecer apoio, permitindo a automatização dos ajustes relativos a execução em cada plataforma, abstraindo este detalhe do desenvolvedor, que, aliado ao código-fonte único garante maior produtividade.

Assim, além dos recursos tradicionais, como codificação da interface, *drag and drop* (arrastar e soltar), ajuste posterior (manutenção) e visualização em tempo real, também se verifica o quanto automatizado é o ajuste da interface para as plataformas diferentes.

Os quesitos comparados e avaliados serão:

- Codificação da *interface* (comandos ou visual);

- É possível arrastar componentes para a *interface* (sim/não);

- Manutenção é trivial;

- Visualização da *interface* em tempo de projeto (sim/não);

- Efetua ajuste automatizado da *interface* para as plataformas (sim/não/parcialmente);

- Necessita de ajuste manual (via código) para determinada plataforma (sim/não, qual).

## 4.5 Testes do Aplicativo

Uma das etapas importantes é o teste do aplicativo móvel, que pode ser realizado tanto com a utilização de um emulador ou simulador de dispositivo como no equipamento físico propriamente dito. Por questões relacionadas aos custos ou mesmo dificuldades técnicas, não é possível testar as aplicações móveis sempre com o uso de dispositivos físicos. Desta forma, os simuladores e emuladores têm grande importância.

No entanto, nem todas as ferramentas possuem um emulador ou simulador próprios. Além disso, os emuladores podem ter características muito diferentes, dependendo da plataforma. Por isto, o que deve ser considerado é a facilidade em, efetivamente, fazer o teste a partir da ferramenta de desenvolvimento. Este processo deve ser rápido e com poucos passos, evitando-se o desperdício de tempo na realização de tarefas que não são o objetivo no processo de teste. Assim, consideram-se pontos relevantes: emulador/simulador integrado à ferramenta, *deploy* automático para o emulador, comandos visuais ou não e tempo de montagem do aplicativo.

Ainda em relação ao teste em dispositivo físico ele é mais sensível, dependendo da plataforma, pois poderá haver dependência, inclusive de *hardware*. Por este motivo, apesar de detalhar o procedimento para cada plataforma, não será considerado efetivamente como relevante na escolha de uma ou outra ferramenta.

Com isso, os pontos de comparação são:

- Existe emulador integrado à ferramenta (sim/não/detalhamento por plataforma);

- *Deploy* de teste é automatizado (sim/não/parcialmente);

- Os comandos de *deploy* podem ser executados visualmente na interface da ferramenta;

- Tempo de montagem do aplicativo para execução de teste.

## 4.6 Distribuição

A distribuição dos aplicativos móveis é realizada e centralizada na loja de cada plataforma, ainda assim, algumas ferramentas possuem procedimentos específicos, que devem ser executados, para a geração do aplicativo propriamente dito, que será então, disponibilizado na loja de cada plataforma. Aliado a isto, por segurança e controles existentes nas lojas de aplicativos, as plataformas podem exigir que o aplicativo seja "empacotado" com a identificação do desenvolvedor. Este processo é feito com o apoio da ferramenta, que deve garantir os subsídios mínimos para que o desenvolvedor tenha menor esforço para executá-lo, tanto na geração do aplicativo para uma plataforma, bem como, quando for necessário, em pouco tempo, efetuar a geração para todas as plataformas. Neste quesito é avaliado o tempo e a quantidade de comandos (visuais ou não) para executar a geração do aplicativo.

## 4.7 Desempenho do aplicativo móvel

Considerando as diferenças no processo de desenvolvimento multiplataforma para cada ferramenta, o desempenho do aplicativo móvel pode transparecer, sendo mais ou menos rápido. Pelo fato dos usuários estarem já habituados aos aplicativos que têm um desempenho aceitável, ou seja, que não requeira muita espera quando for executada alguma ação, é importante que este seja o comportamento do aplicativo, bem como não existam grandes discrepâncias na execução de operações comuns. Desta forma, serão feitas medições de tempo, correspondente à execução das seguintes operações no aplicativo:

- Abertura do aplicativo;

- Inclusão de registro no banco de dados local;

- Consumo de *web-service*;

- Envio de e-mail.

## 4.8 Avaliação Geral

O processo de desenvolvimento de um aplicativo móvel não está restrito aos pontos acima. Desta forma, após a comparação e análise dos mesmos, faz-se necessário incluir detalhes e experiências, além de estatísticas que contribuam para que o comparativo possa dar um vislumbre real de utilizar uma das ferramentas, garantindo maior ciência e clareza na escolha. Na possibilidade de identificar uma ferramenta que se destaque muito em relação as demais, isto será evidenciado, indicando que é a melhor alternativa.

# 5 TRABALHO DESENVOLVIDO

Este Capítulo descreve o trabalho desenvolvido, através da instalação e utilização das ferramentas de desenvolvimento multiplataforma Xamarin, TotalCross, Cordova, PhoneGap e Ionic. Nesse contexto é feita a análise e comparação das ferramentas conforme itens descritos no Capítulo 4, correspondente a metodologia.

## 5.1 Ferramentas e versões utilizadas

No desenvolvimento deste trabalho foram utilizadas as IDE's e recursos de apoio descritos a seguir, baseando-se, principalmente na documentação de cada ferramenta, tentando explorar ao máximo os recursos de cada uma que visam a qualidade e agilidade na produção de aplicativos.

**Xamarin**

Microsoft Visual Studio Community 2015, versão 14.0.25431.01 Update 3;

Xamarin versão 4.2.0.719, integrado ao Visual Studio;

Máquina Virtual Mac OS X Sierra, versão 10.12.1;

XCode, versão 8.1;

Xamarin Studio para Mac, versão 10.2.0.4.

**TotalCross**

TotalCross 3, versão 3.1;

Eclipse Neon 1a Release, versão 4.6.1;

Oracle SDK Java 8, Atualização 111, versão 1.8.0_111-b14.

**Cordova**

Cordova 6, versão 6.4.0.

**PhoneGap**

PhoneGap 6, versão 6.4.2;

Cordova 6, versão 6.4.0.

**Ionic**

Ionic 2, versão 2.1.12;

Cordova 6, versão 6.4.0.

## 5.2 Equipamentos utilizados

Durante todo o desenvolvimento deste trabalho foram utilizados os seguintes equipamentos e sistemas, garantindo a imparcialidade da avaliação e comparação.

### Computador

A máquina utilizada para o desenvolvimento dos aplicativos possui as seguintes características:

- Marca/Modelo: Notebook Asus S46C;

- Processador: Intel® Core™ i7-3537U CPU @2.00GHz 2.5 GHz;

- Memória Ram: 8 GB;

- Disco Rígido: HDD 1TB + SSD 24GB;

- Sistema Operacional: Windows 10 Pro 64 bits.

Pelas características do Sistema iOS e da ferramenta Xamarin, também foi necessário configurar uma máquina virtual com as seguintes definições:

- Processador virtualizado: Dois Núcleos;

- Memória Ram Compartilhada: 4GB;

- Disco Rígido: HDD 80GB;

- Sistema Operacional: Mac OS X Sierra 10.12.1.

### *Smartphones*

Além dos emuladores e simuladores o aplicativo também foi instalado nos *smartphones*:

- iPhone 4 com Sistema Operacional iOS 7.1.2;

- Microsoft Lumia 640 LTE com Sistema Operacional Windows 10 Mobile 10.0.14393;

- Sony Xperia Walkman com Sistema Operacional Android versão 4.0.3.

## 5.3 Ferramentas de Desenvolvimento Avaliadas

Devido à limitação no escopo deste trabalho, todas as ferramentas foram instaladas, configuradas e utilizados no Sistema Operacional Windows 10 Pro 64 bits. Desta forma todas

as observações e considerações levam em conta a realidade neste sistema.

### 5.3.1 Xamarin

O processo de instalação e configuração da ferramenta Xamarin foi bastante simplificado, tanto em relação à instalação independente do Xamarin Studio, como, na integrada com o Microsoft Visual Studio. Nos dois casos, o processo de instalação e configuração foi realizado com o apoio de um assistente, onde a maioria das ações consistiu em apenas aceitar a opção padrão e passar ao próximo passo. A forma de disponibilização do pacote de instalação da ferramenta é através de *download* no *site* da Xamarin.

Apesar do processo ser bastante simplificado, o tamanho dos arquivos de instalação que foram baixados da internet era grande, fazendo com que, dependendo da velocidade da internet, fosse um dos passos mais demorados. Ao optar pela IDE Visual Studio, o arquivo de imagem do DVD de instalação possui o tamanho de 3,69GB. Caso a escolha seja pelo Xamarin Studio, os arquivos a serem baixados somam 1,67 GB.

A configuração foi realizada automaticamente pelo assistente de instalação, de modo que, após sua conclusão, a ferramenta já está pronta para utilização, requerendo apenas o *download* em separado do SDK do Android, utilizando o próprio gerenciador de SDK disponibilizado pelo Google que já foi baixado e instalado de maneira integrada ao Xamarin.

A ferramenta Xamarin é distribuída sob quatro formas de licenciamento, sendo elas: Xamarin Studio Community, Visual Studio Community, Visual Studio Professional e Visual Studio Enterprise. As primeiras duas são gratuitas, enquanto as duas últimas são pagas. A principal diferença entre as versões gratuitas e pagas estão mais relacionadas ao suporte e a quantidade de desenvolvedores envolvidos. Especificamente, não há limitações de funcionalidades disponíveis para o desenvolvimento multiplataforma. Obviamente existem alguns pontos extras, nas versões pagas, porém, trata-se de recursos independentes da ferramenta, providos por sua mantenedora. Os valores correspondentes às versões pagas não estão disponíveis no site oficial, sendo necessário preencher um formulário para obter mais informações sobre estas formas de licenciamento.

Pode-se dizer que o Xamarin possui, não apenas um ambiente de desenvolvimento integrado nativo, mas na verdade dois. Historicamente, o Xamarin sempre disponibilizou seu próprio IDE, chamado de Xamarin Studio, mas também, esteve disponível na forma de um *plugin* no IDE Visual Studio. Com a consolidação desta integração e, recentemente, com a

aquisição do Xamarin, pela Microsoft, desenvolvedora do Visual Studio, o Xamarin tornou-se um componente obrigatório e, quase, nativo da ferramenta.

Neste trabalho, a IDE utilizada foi o Visual Studio Community, visto que, apesar da licença gratuita, não haveria limitações de funcionalidades. Além disso, possui recursos extras que facilitam o desenvolvimento, sendo também uma ferramenta muito utilizada no mercado e reconhecida pelos seus diferenciais no desenvolvimento de aplicações da plataforma .NET.

A linguagem de programação utilizada no Xamarin é a C#. Todas as classes, recursos e pacotes de bibliotecas externas da linguagem C# e da plataforma .NET estão disponíveis, assim como as classes implementadas pela Xamarin, voltadas especificamente ao desenvolvimento multiplataforma.

Durante o desenvolvimento ficou evidente que, para os desenvolvedores com conhecimento prévio da linguagem, o trabalho é mais produtivo, pois o aprendizado fica restrito à utilização de classes e funcionalidades específicas, disponibilizadas pela Xamarin. Mesmo assim, a forma de utilização segue o padrão da linguagem C#, que é destaque na plataforma .NET.

Aliado a isso, também existe a característica de que a principal IDE, utilizada para desenvolvimento com a linguagem C# também é o Visual Studio. Desta maneira, para os desenvolvedores com prévio conhecimento da linguagem, torna-se um caminho muito natural a adoção da mesma IDE para trabalhar com Xamarin. Com o uso da IDE Visual Studio, todas as funcionalidades que apoiam a escrita de código estão presentes, assim: sugestão de código baseado no contexto, *auto complete*, validação de código em tempo de digitação. Também está disponível toda a vasta quantidade de *plugins*, classes e métodos da plataforma .NET, o que permite, para os já familiarizados com o ambiente, maior produtividade.

Por outro lado, o formato peculiar de desenvolvimento multiplataforma do Xamarin acaba criando uma preocupação e, até, trabalho extra, por parte do desenvolvedor, de modo a realmente garantir a unicidade de código-fonte. Isso ocorre, porque ao criar um novo projeto multiplataforma no Xamarin, além do projeto principal (*shared* ou *portable*, de acordo com o que foi selecionado), são criados subprojetos, um para cada plataforma destino do aplicativo em desenvolvimento. A partir disso, é necessário trabalhar de maneira que todo o desenvolvimento de código seja realizado no projeto principal e compartilhado ou portado para os subprojetos quando o aplicativo for testado ou executado nos dispositivos.

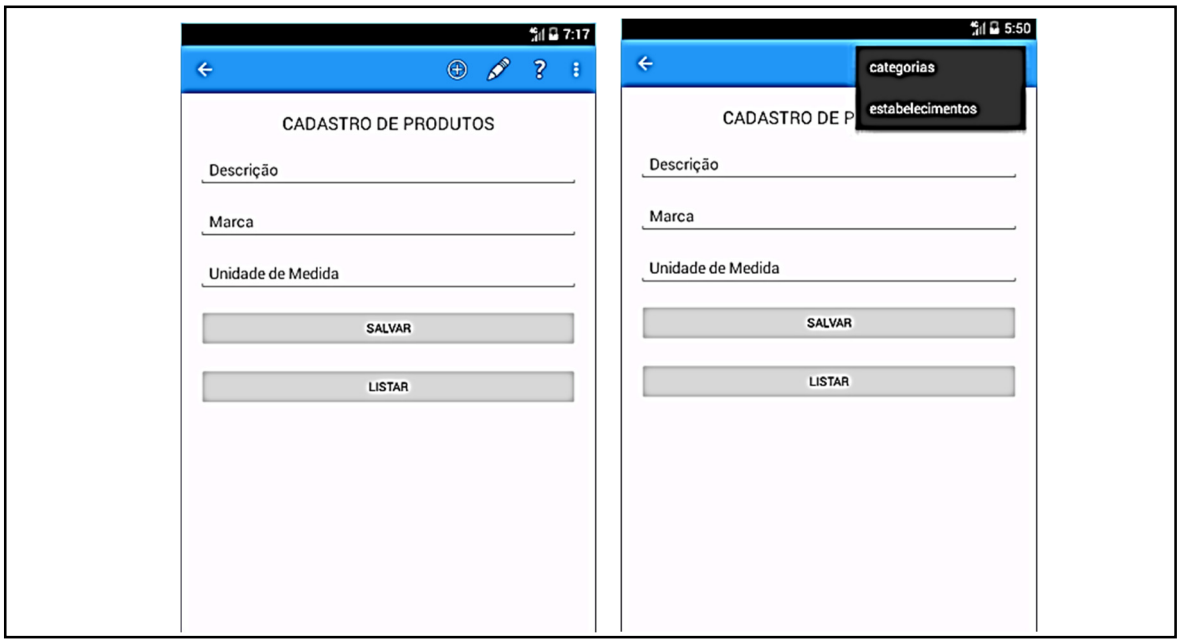
Este esforço de manter o código concentrado no projeto principal (*shared* ou *portable*)

acabou estendendo o desenvolvimento de determinadas funcionalidades, para as quais havia peculiaridades de plataforma que impediam a utilização de código totalmente compartilhado. Especificamente na utilização do SQLite e do acesso à câmera do dispositivo, foi necessário efetuar codificação específica no subprojeto de cada plataforma de modo a obter a conexão com o arquivo do banco de dados ou acessar e tratar arquivos de fotos. Evidentemente, buscou-se manter o máximo de código possível no formato compartilhado, para evitar a fragmentação nos subprojetos de cada sistema móvel.

O esforço em manter o código unificado mostrou-se o maior causador no alto tempo de desenvolvimento, já que, do total de 7 horas de desenvolvimento de uma tela de cadastro, foram destinadas 4 horas apenas para definir e ajustar o funcionamento da persistência de dados, de modo a mantê-la razoavelmente independente de plataforma e concentrada no projeto principal.

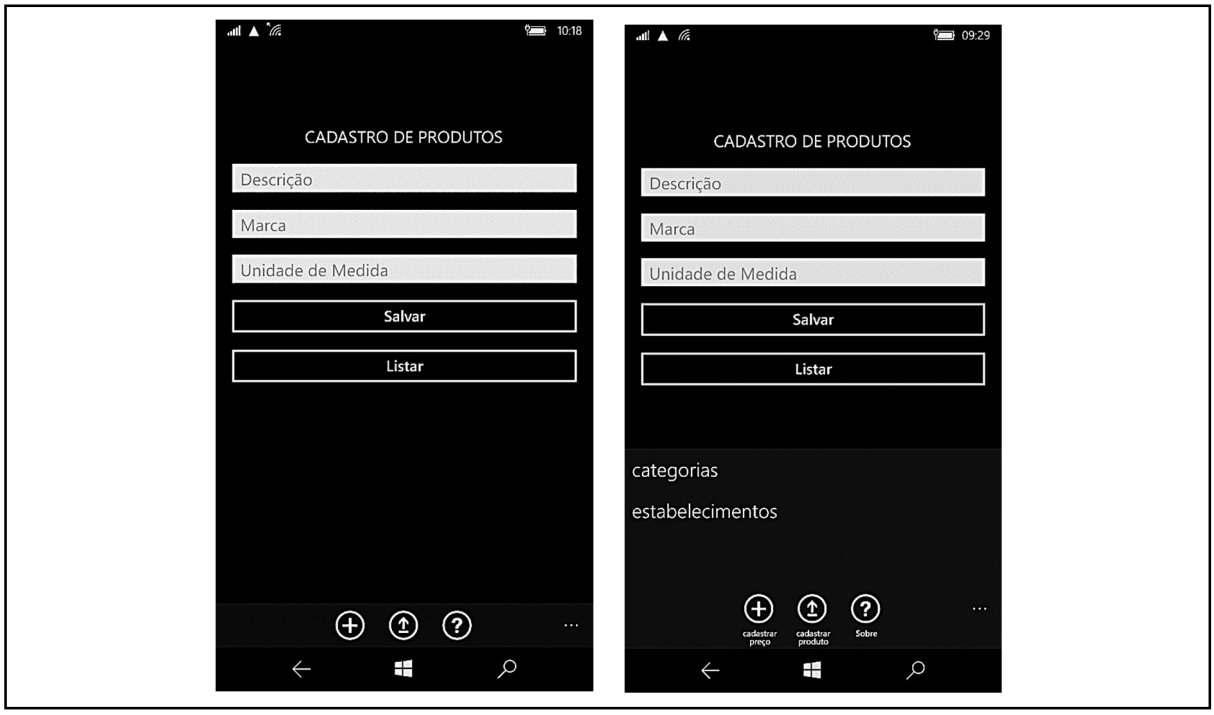
O desenho da interface foi realizado através de código XAML. Este formato de escrita da interface permite o compartilhamento completo dentre as plataformas. Para complementar, a ferramenta também efetua, de forma automática, os ajustes devidos para cada sistema operacional, deixando o visual extremamente parecido com aplicativos produzidos de forma nativa, utilizando as ferramentas padrão das plataformas. É possível visualizar a adequação automatizada do aplicativo em cada plataforma, principalmente do menu principal, na Figura 5, executando no emulador do Android e, na Figura 6, executando no emulador do Windows Phone.

**Figura 5: Aplicativo ComparaPreco executando na plataforma Android**




Fonte: Elaborada pelo autor.

**Figura 6: Aplicativo ComparaPreco executando na plataforma Windows Phone**




Fonte: Elaborada pelo autor.

Vale ressaltar que o código-fonte da interface ficou separado do código-fonte destinado ao controle de execução do aplicativo. Esta separação, como padrão na linguagem C#, foi feita fisicamente, pela própria ferramenta e linguagem, na forma de gravação, executada em arquivos separados.

Durante a escrita da interface foi necessário tratar alguns pontos de forma específica para cada plataforma. Apesar da adequação automática de menus, botões e outros componentes para o sistema operacional onde o aplicativo está executando, foi preciso definir manualmente o arquivo de origem que continha cada ícone de botões, menus ou outros componentes da interface. Na Figura 7 é possível visualizar trecho de código XAML da interface onde há tratamento específico para cada plataforma, quando é utilizada a *tag* `OnPlatform`, indicando o caminho específico do arquivo de imagem em cada plataforma.

**Figura 7: Código da linguagem XAML com uso da tag *OnPlatform***

```
  <ContentPage.ToolbarItems>
    <ToolbarItem Text="cadastrar preço" Order="Primary"
Clicked="AbrirCadastroPreco">
      <ToolbarItem.Icon>
        <OnPlatform x:TypeArguments="FileImageSource"
                    iOS="edit.png"
                    Android="Resources/drawable/ic_menu_btn_add.png"
                    WinPhone="Images/Dark/add.png" />
      </ToolbarItem.Icon>
```

Fonte: Elaborada pelo autor.

Infelizmente, não foi possível verificar, em tempo de projeto, como seria a visualização da *interface* construída, sendo necessário executar o aplicativo nos emuladores ou dispositivos físicos para verificar a necessidade de ajustes, ou mesmo, apenas obter o resultado visual dos comandos que definem as telas. Também não foi possível arrastar componentes para a *interface*, sendo obrigatório a digitação dos códigos correspondentes a quaisquer botões, controles e outros itens de tela que sejam necessários.

Sempre que houve a necessidade de eventuais manutenções na *interface*, o processo foi banal, consistindo apenas em editar e alterar o código correspondente a tela a ser ajustada. Não ocorreram problemas neste processo, nem mesmo, complicadores na tarefa, durante ou depois da alteração.

#### 5.3.1.1 Aprendizado

Em relação ao aprendizado necessário para o desenvolvimento, utilizando a ferramenta Xamarin, estão: os conhecimentos na IDE Visual Studio, na linguagem C# e na linguagem XAML, necessária para a definição de *interface* multiplataforma. Nestes quesitos, a IDE e a linguagem C# destacaram-se como pontos positivos, pois a IDE é universalmente conhecida e C# tem origem na linguagem C, predecessora das linguagens de programação mais utilizadas no mundo atualmente, nas quais está a própria C#. A linguagem XAML não trouxe dificuldades na utilização, porém, requereu um tempo de adaptação em seu uso e, juntamente com o conceito de desenvolvimento multiplataforma, representou o maior desafio na programação.

#### 5.3.1.2 Desenvolvimento de funcionalidades específicas

Para efetuar a gravação local de dados, ou seja, diretamente no dispositivo, foi utilizado o SQLite, por ser uma das opções muito populares entre os desenvolvedores. Além disso, ela é

totalmente compatível com as três plataformas. A implementação foi realizada com as classes disponíveis no pacote Sqlite-net-pcl, que dentre outras possibilidades, foi o que permitiu unificar a maior parte do código-fonte necessário para executar as operações de banco de dados.

Dentre as operações realizadas com o SQLite, a que demandou maior trabalho foi a tarefa de efetuar a conexão com o banco de dados que, neste caso, é a abertura do arquivo, no qual as informações serão registradas ou recuperadas. Considerando que o sistema de arquivos de cada plataforma é diferente, não apenas em relação as pastas e arquivos disponíveis, mas também, a estrutura e forma de acesso, não houve a possibilidade de unificar a localização e recuperação do arquivo de dados do SQLite utilizando código compartilhado. Na Figura 8 é possível visualizar o tratamento específico que foi necessário para obter a localização do arquivo do banco de dados SQLite na plataforma Android.

**Figura 8: Código da linguagem C# específico para plataforma Android**

```
private static string GetDatabasePath()
{
    const string sqliteFilename = "comparapreco.db3";

    string documentsPath =
System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
    var path = Path.Combine(documentsPath, sqliteFilename);
    return path;
}
```

Fonte: Elaborada pelo autor.

Levando em consideração a tentativa de minimizar a existência de código específico por plataforma, todas as operações foram implementadas no projeto compartilhado, onde a definição de uma classe do tipo Interface permitiu desenvolver o código correspondente em cada subprojeto fazendo o acesso ao arquivo, respeitando as funções e limitações de cada sistema operacional.

Assim como ocorreu no uso do SQLite, o acesso à câmera através de um código único e compartilhado não foi possível. Devido as características das diferentes plataformas tornou-se necessário implementar praticamente todo o código nos projetos específicos de cada sistema operacional, mesmo com a utilização das classes do pacote `Xamarin.mobile` que, pelo menos, possibilitaram simplificar a codificação do acesso à câmera do dispositivo. Aliás, a única tarefa que pode ser unificada no código compartilhado entre todas as plataformas foi o acesso à câmera. Na Figura 9 pode-se ver trecho de código compartilhado, único para as três plataformas, cujo uso permitiu acessar a câmera do dispositivo e capturar uma foto em um

diretório determinado e com um nome definido para posterior recuperação.

**Figura 9: Código compartilhado da linguagem C# para acesso ao recurso Câmera**

```
try
{
    MediaFile file = await picker.TakePhotoAsync(new StoreCameraMediaOptions
    {
        Name = "produto_01.jpg",
        Directory = "FotosProdutos"
    });
    await DisplayAlert("Foto Salva", file.Path, "OK");
} catch (OperationCanceledException)
{
    await DisplayAlert("Cancelamento", "Foto cancelada!", "OK");
}
```

Fonte: Elaborada pelo autor.

Apesar disso, as manipulações, ou mesmo a exibição das fotos capturadas, exigiram codificação específica por plataforma, ainda que, conforme mencionado, tenham sido utilizadas classes do pacote `Xamarin.mobile`, com o intuito inicial de manter o código unificado.

A funcionalidade de consumir um *web-service* foi totalmente codificada no projeto compartilhado e, além de compartilhar o mesmo código dentre as plataformas, foi extremamente trivial em relação à implementação. O poder da linguagem C#, em relação à desserialização do JSON recebido, permitiu resolver tudo em poucas linhas de código e, apenas utilizando a biblioteca `Newtonsoft.Json`, disponível na plataforma .NET.

Esse processo simplista deveu-se, em grande parte, às características da linguagem, pelas quais os objetos JSON retornados do consumo do serviço foram transformados diretamente em atributos de classes, conhecidos pela aplicação. A seguir, foi possível manipular livremente os dados, utilizando os próprios métodos das classes.

O compartilhamento de informações, que permite efetuar uma chamada a outro aplicativo externo ou de terceiros, encaminhando informações a ele, também foi de fácil implementação no Xamarin. Todo o código para esta funcionalidade específica foi único e compartilhado entre todas as plataformas.

Esta implementação unificada foi possível com a utilização das classes disponíveis no pacote Plugin.Share. As funcionalidades disponibilizadas pelas classes correspondentes permitiram, além do compartilhamento pelo qual o usuário escolhe o aplicativo de terceiro destino das informações, também a abertura do navegador de internet em um *link* específico.

#### 5.3.1.3 Testes com o aplicativo

A partir da IDE Visual Studio foi possível gerenciar e executar os emuladores para as três plataformas, que já estavam disponíveis logo após a conclusão da instalação e configuração inicial. Ainda que, para executar o emulador do iOS, devido às restrições impostas pela Apple, tenha sido necessário manter conexão de rede a uma máquina virtual, executando o sistema operacional Apple MAC OS X Sierra, além de possuir o XCode instalado e configurado juntamente com o Xamarin Studio para MAC. Foram necessários diversos ajustes e configurações, no MAC OS e no Visual Studio, que dispenderam um tempo bem considerável, até ser possível utilizar o emulador, além de manter, obrigatoriamente o controle da versão do Xamarin nos dois ambientes, que deve ser, obrigatoriamente a mesma.

O emulador do Android não tem restrições, sendo o mesmo utilizado pela IDE de desenvolvimento nativo. Já o emulador do Windows Phone faz parte do Visual Studio, pois a mesma IDE é utilizada para o desenvolvimento nativo desta plataforma.

O acesso a qualquer um dos emuladores, tanto para ajustes na configuração, como para a execução, foi feita diretamente na barra de ferramentas do Visual Studio, inclusive para o iOS e Android. Para efetuar o *debug* ou simulação do aplicativo em desenvolvimento, o processo foi simples e consistiu em iniciar o emulador desejado, através do botão correspondente na barra de tarefas do Visual Studio e, após a conclusão de sua inicialização, clicar no botão de execução do aplicativo, no qual, dentre as opções listadas, estão, automaticamente, os emuladores carregados naquele momento.

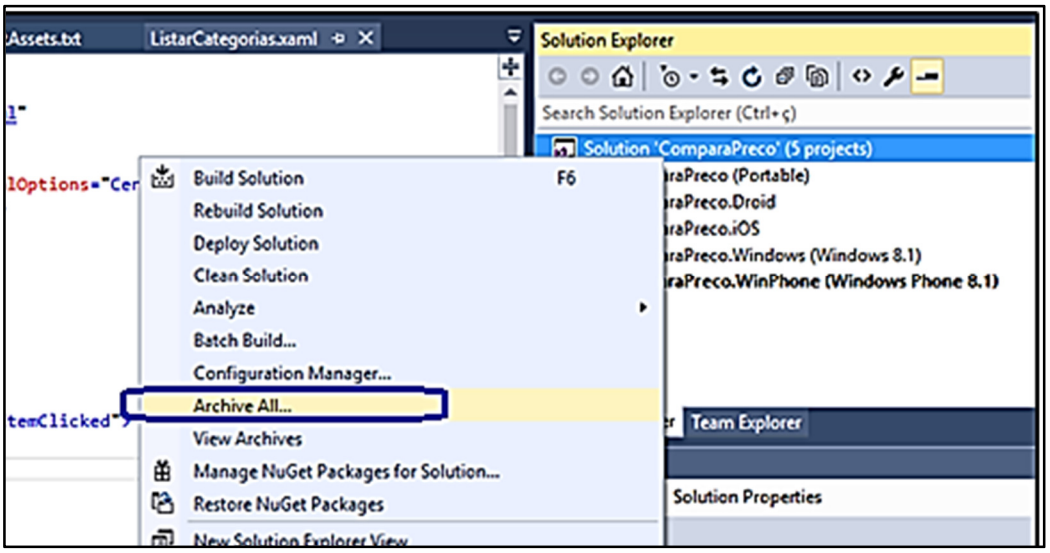
Os emuladores executados a partir do Xamarin atenderam muito bem a necessidade de testar o aplicativo em desenvolvimento, assim como o funcionamento em cada plataforma. Isto se deve ao fato de que, são utilizados os emuladores nativos de cada plataforma, de modo que não poderia haver outros mais apropriados. Apesar disso, o tempo para testar ou executar um aplicativo nos emuladores foi o maior dentre as ferramentas avaliadas.

#### 5.3.1.4 *Deploy* e distribuição

Os recursos da IDE Visual Studio permitem gerar o aplicativo final, para publicação na loja de aplicativos de cada plataforma. Através do comando visual "*Archive*", disponível clicando com o botão direto sobre o projeto a ser publicado foram gerados os arquivos contendo a aplicação para a plataforma correspondente. Também foi possível gerar os arquivos para todas as plataformas contidas no projeto global, clicando com o botão direito sobre o mesmo e

acionando a opção "*Arquive All...*", conforme ilustrado na Figura 10.

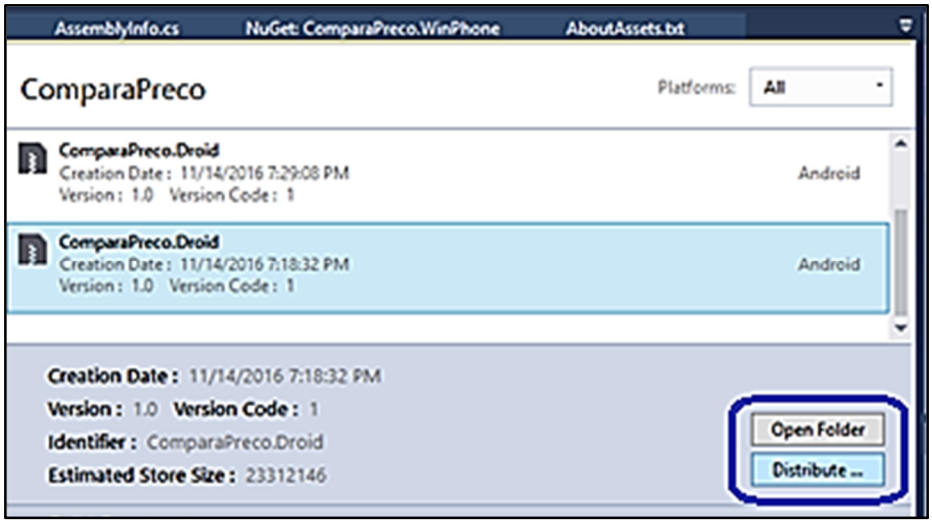
**Figura 10: Opções do Visual Studio para gerar os instaladores do aplicativo**




Fonte: Elaborada pelo autor.

Após a geração dos arquivos contendo o instalador do aplicativo para cada plataforma, também foi possível efetuar a distribuição diretamente a partir da ferramenta, permitindo a integração direta com a loja de aplicativos de cada sistema operacional. A Figura 11 apresenta os botões, "*Open Folder*" e "*Distribute...*" que, respectivamente, permitem acessar a pasta com o arquivo gerado ou então, iniciar a interação com a loja de aplicativos para a publicação do mesmo. Esse é um diferencial perante as demais ferramentas, nas quais foi necessário executar estes passos manualmente.

**Figura 11: Comandos visuais para execução de *deploy***




Fonte: Elaborada pelo autor.

Mesmo com este processo de distribuição automatizado, a documentação do Xamarin possui instruções para apoiar os desenvolvedores nesta tarefa, visando deixar o procedimento ainda mais rápido e com maior possibilidade de sucesso.

### 5.3.2 TotalCross

A utilização do TotalCross se dá através da IDE Eclipse. Portanto, foi necessário atender alguns pré-requisitos para conseguir efetuar a sua instalação e configuração. O primeiro passo foi o *download* e instalação do SDK Java, realizado com o apoio do assistente, o que foi relativamente simples e rápido. Em seguida, foi necessário efetuar o *download* da IDE Eclipse, que não precisou ser instalada, apenas extraído o conteúdo do arquivo comprimido em uma pasta determinada pelo próprio utilizador. Na primeira execução da IDE, ela efetuou a localização do SDK Java, necessário à sua execução, utilizando para tanto as variáveis de ambiente do sistema operacional Windows, previamente ajustadas pelo assistente de instalação do SDK Java.

Depois da instalação dos pré-requisitos, a instalação do TotalCross foi trivial, sendo preciso baixar o instalador a partir do *site* oficial e executá-lo. O instalador do TotalCross é um arquivo compactado auto extraível, que, quando executado, solicitou a pasta destino da ferramenta TotalCross.

O SDK Java, a IDE Eclipse e o TotalCross são disponibilizados em pacotes para serem baixados a partir dos *sites* oficiais de cada um. O tamanho dos pacotes é relativamente pequeno, sendo composto por 189MB do SDK Java, 301MB do Eclipse e ~43MB do TotalCross. No total, os três itens totalizam 533MB.

Para utilizar o TotalCross no Eclipse havia duas possibilidades: a primeira era baixar um projeto de exemplo, o qual já possuía os apontamentos para as bibliotecas dependentes do TotalCross na localização padrão, a segunda era criar um novo projeto Java e fazer os apontamentos das bibliotecas dependentes do TotalCross manualmente. Em ambas as opções não há, absolutamente, necessidade de procedimentos complexos. Além disso, a documentação da ferramenta fornece tutorias de apoio para executar a configuração. Foram realizadas as duas alternativas e as operações transcorreram normalmente, apenas com o inconveniente de importar manualmente as bibliotecas do TotalCrosso no projeto do Eclipse na segunda opção.

Recentemente, o formato de licenciamento do TotalCross foi alterado. Anteriormente pago, ele passou a ser gratuito. Desta forma, ao efetuar o *download* de seu instalador no *site*

oficial, foi necessário preencher algumas informações, incluindo o e-mail do utilizador, para o qual foi enviada a chave da licença, necessária tanto para testar as aplicações em simuladores, bem como para efetuar a geração do aplicativo móvel que poderia ser disponibilizado nas lojas de aplicativos de cada sistema.

A ferramenta TotalCross não possui um ambiente de desenvolvimento integrado nativo, assim como não requer que seja utilizado algum em específico. Ainda assim, os tutoriais presentes na documentação da ferramenta apontam na utilização do IDE Eclipse que já é amplamente difundida na comunidade de desenvolvedores Java.

Apesar de ter sido necessário configurar manualmente a importação das bibliotecas do TotalCross na IDE Eclipse, isto não afetou a rapidez de configuração, mesmo quando comparado com a ferramenta Xamarin, que, além de possuir uma IDE própria, também dispunha de um configurador automatizado.

Pode-se dizer que a adoção do Eclipse para a TotalCross, assim como, o Visual Studio para a Xamarin teve impacto positivo, devido a prévia familiaridade dos desenvolvedores com as IDE's, garantindo uma boa produtividade e menor curva de aprendizado.

A linguagem de programação no TotalCross é o Java. Todos os recursos e classes nativas da linguagem, assim como, as classes e funcionalidades implementadas pela TotalCross, estavam disponíveis para o desenvolvimento multiplataforma.

Durante o desenvolvimento com TotalCross ficou evidente que a alta produtividade é certa para quem já desenvolveu com a linguagem Java, sendo preciso apenas lidar com o aprendizado de classes específicas do TotalCross, adequadas para a implementação multiplataforma.

Utilizando a IDE Eclipse, o desenvolvimento do aplicativo ComparaPreco na ferramenta TotalCross mostrou-se extremamente produtiva. A documentação da ferramenta é padrão, porém, o apoio da IDE e a familiaridade com a linguagem de programação Java fizeram com que o desenvolvimento de um aplicativo móvel fosse trivial, mesmo tendo sido realizado na ferramenta pela primeira vez.

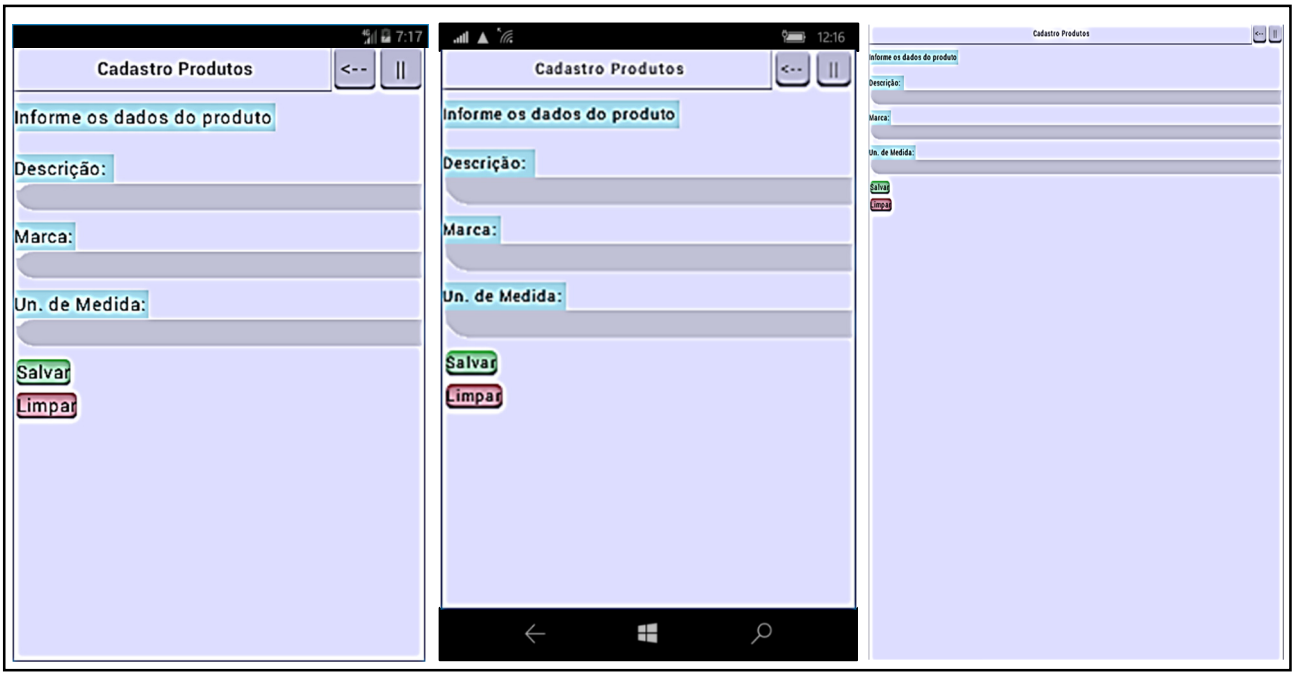
Como ponto negativo é importante salientar a mistura de código-fonte de rotinas e regras com o código da interface. Apesar da orientação a objetos da linguagem Java, não foi possível, por exemplo, separar o código-fonte da interface de uma tela do código das rotinas associadas a ela em diferentes arquivos ou classes. Neste contexto, o esforço foi manter uma distância segura dos dois tipos de código, viabilizando futuras manutenções ou correções de *bugs* com

menor dispêndio de tempo.

Considerando a grande diversidade de classes específicas fornecidas pelo TotalCross e todo o legado compreendido pela linguagem Java, não houve funcionalidades com grande complexidade de desenvolvimento. Tendo tomado como exemplo o tempo para implementar um cadastro simples, ele foi consideravelmente menor do que quando realizado nas outras ferramentas.

Através de classes específicas implementadas e disponibilizadas pela ferramenta escreveu-se a interface do aplicativo. Não houve ajustes automáticos, por parte da ferramenta, que permitisse à interface moldar-se a uma ou outra plataforma durante a execução. Ainda assim, foi possível aplicar alguns efeitos visuais específicos, de um ou outro sistema operacional móvel, porém, ao fazê-lo, a alteração foi universal, ou seja, ao executar a aplicação, o visual era o mesmo em todos os sistemas, independentemente da plataforma em que estava executando, conforme pode ser visto na Figura 12.

**Figura 12: Aplicativo executando nos simuladores de Android, Windows e iOS**




Fonte: Elaborada pelo autor.

A escrita foi totalmente unificada para todas as plataformas e realizada com a mesma linguagem Java do código-fonte das rotinas do aplicativo. Além disso, os códigos ficaram agrupados em classes, onde misturaram-se definição de interface e rotinas da aplicação. Apesar de ter permitido customização específica para uma ou outra plataforma, ela serviu apenas para adotar um padrão visual específico de um sistema operacional. Na figura 13 é possível

visualizar trecho de código de uma classe, contendo a definição de uma interface.

**Figura 13: Código de interface multiplataforma com a linguagem Java**

```
bar = new Bar("Cadastro Produtos");
       bar.titleAlign = CENTER;
Image ic = Resources.menu.getCopy();
       ic.applyColor2(0); // change button to black
btVoltar = new Button("<--");
       bar.addControl(btVoltar);
       bar.ignoreInsets = true;
       add(bar, LEFT,TOP,FILL,PREFERRED);

c = new ScrollContainer(false, true);
       c.add(new Label("Informe os dados do produto"),LEFT,TOP+50);

       c.add(new Label("Descrição: "), LEFT,AFTER+100);
       c.add(edNome = new Edit(),LEFT,AFTER);

       c.add(new Label("Marca:"), LEFT,AFTER+50);
       c.add(edMarca = new Edit(),LEFT,AFTER);

       c.add(new Label("Un. de Medida:"), LEFT,AFTER+50);
       c.add(edMedida = new Edit(),LEFT,AFTER);

       c.add(btSalvar = new Button("Salvar"), LEFT, AFTER+50);
       c.add(btLimpar  = new Button("Limpar"),  LEFT, AFTER+20);

btSalvar.setBackColor(Color.GREEN);
btLimpar.setBackColor(Color.RED);

c.setBackColor(0xDDDDFF);
       add(c, LEFT, AFTER, FILL, FILL);
       produtosData = new ProdutosData();
```

Fonte: Elaborado pelo autor.

### 5.3.2.1 Aprendizado

A utilização da IDE Eclipse e da linguagem Java foi fácil, em parte, devido ao prévio conhecimento sobre as mesmas. A IDE, apesar de modular e com inúmeros *plugins*, conta com grande comunidade ativa, que atua também apoiando a resolução de dúvidas. Por outro lado, a linguagem Java é extremamente conhecida e também deriva da linguagem C. Sendo assim, elas não representam uma barreira, mesmo para quem ainda não as conhece. Pelas características da ferramenta TotalCross, que forneceu classes permitindo o desenvolvimento de funcionalidades voltadas aos dispositivos móveis, o desenvolvimento foi praticamente o mesmo que seria utilizado na implementação de qualquer aplicação Java.

A TotalCross permitiu uma excelente produtividade e agilidade no desenvolvimento, não apenas devido à linguagem Java, mas também, em grande parte, pela pouca customização relacionada a adequação da interface para as diferentes plataformas, sendo, o aplicativo resultante, praticamente uma caixa fechada, executando em uma máquina virtual no dispositivo móvel.

#### 5.3.2.2 Desenvolvimento de funcionalidades específicas

Devido ao versátil tratamento de conexões com banco de dados do Java, possibilitado pelo uso do recurso de JDBC, que se conecta praticamente a qualquer banco de dados conhecido, a persistência de dados foi implementada de forma totalmente universal, com um único código para todas as plataformas.

Por ser através do JDBC, além do benefício de ser um código multiplataforma, o conhecimento prévio deste recurso possibilitou agilidade no desenvolvimento desta funcionalidade.

A ferramenta TotalCross disponibiliza a classe `Camera` para a comunicação com o recurso câmera do dispositivo móvel onde o aplicativo está executando. Esta classe é totalmente independente de plataforma, de modo que não é necessário nenhum ajuste por parte do programador, para garantir o funcionamento em um ou outro sistema operacional.

Os métodos disponibilizados pela classe `Camera` são limitados e resumem-se a disparar a captura de uma foto na câmera principal do dispositivo e retornar o caminho e nome completo do arquivo armazenado, contendo a imagem capturada. Também é possível efetuar a gravação de vídeos, indicando o momento inicial e final da gravação, sendo que, ao final, também é retornado um valor do tipo *String* que contém a localização do vídeo capturado.

Efetivamente, a implementação do recurso consistiu na utilização da classe Camera, tendo feito uso do método que faz a captura da foto e a armazena, retornando a *String* com sua localização. Apesar do desenvolvimento ter sido corriqueiro, a funcionalidade em si pareceu carecer de muitos recursos e destoa bastante do resultado das demais ferramentas.

O TotalCross dispõe de diversas classes para trabalhar com *web-services*, especificamente, nos *packages* `Totalcross.json` e `Totalcross.json.zip`, que fazem não apenas a deserialização dos dados JSON em objetos JSON, mas também a conexão *http* com o servidor que disponibiliza os dados. As classes fornecem métodos de alto nível, o que permite abstrair grande parte das manipulações que seriam necessárias, caso fossem

utilizadas as funções nativas da linguagem Java neste processo.

A implementação foi independente de plataforma, com código unificado e, apesar de terem sido utilizadas classes específicas do TotalCross, o processo seguiu os padrões da linguagem Java no desenvolvimento deste tipo de funcionalidade.

No desenvolvimento do aplicativo móvel com TotalCross não foi possível efetuar o compartilhamento de informações de modo tradicional, ou seja, da mesma forma que ocorre nos aplicativos nativos das plataformas, nos quais é possível enviar dados a um aplicativo terceiro. Não foi localizada na documentação ou, mesmo nas classes da ferramenta, recursos que pudessem executar uma chamada de compartilhamento a outro aplicativo, encaminhando informações.

Ainda assim, foi possível implementar o envio de informações por e-mail, porém, de maneira restrita, pois a transmissão do e-mail teve que ser realizada diretamente no aplicativo, não havendo a possibilidade de efetuar uma chamada ao aplicativo de envio de e-mails padrão do sistema.

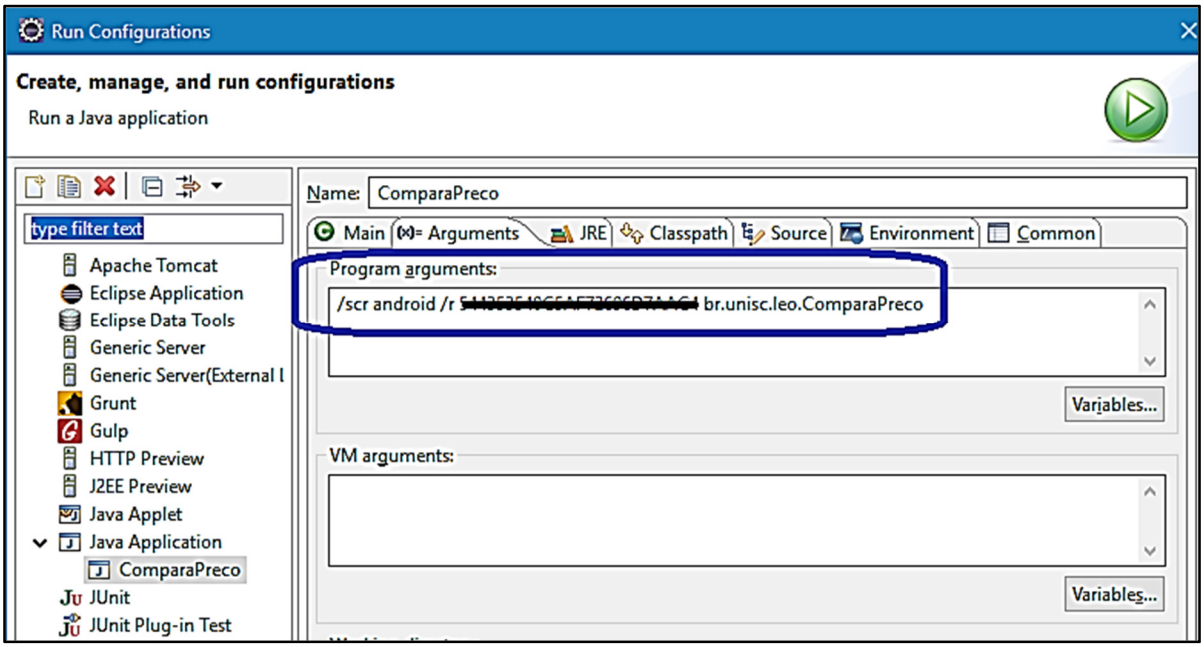
A codificação da funcionalidade de envio de e-mail foi feita utilizando classes do *package* `Totalcross.net.mail`, cujos métodos são extremamente semelhantes aos existentes nas classes nativas do Java que permitem a implementação desta mesma rotina em outros cenários, como o de uma aplicação *web*, por exemplo.

#### 5.3.2.3 Testes com o aplicativo

Os testes do aplicativo desenvolvido com TotalCross foram executados através de um simulador próprio, embarcado na ferramenta. O simulador, na verdade é uma classe especial da ferramenta, a `totalcross.Launcher` que simula a máquina virtual, onde o aplicativo é embarcado e instalado, posteriormente, no dispositivo físico.

Para efetuar a execução do aplicativo no simulador foi necessário apenas ajustar a configuração de execução do aplicativo no Eclipse, apontando a classe `totalcross.Launcher` como a classe principal e passando alguns parâmetros mínimos, como a plataforma, a chave da licença e a classe principal do aplicativo que está sendo simulado. Para simular a execução na plataforma Android, por exemplo, foi definida a seguinte parametrização: `/scr android /r chave_licença_aqui br.unisc.leo.ComparaPreco`, conforme demonstrado na Figura 14.

**Figura 14: Configuração da simulação na IDE Eclipse**




Fonte: Elaborada pelo autor.

Mesmo tendo realizado os testes, alterando o parâmetro correspondente à plataforma que estava sendo simulada, houve pouca diferenciação na execução do aplicativo e na exibição da interface. Ainda que esta limitação esteja, em parte, relacionada ao processo de desenvolvimento do TotalCross, o simulador não retrata de maneira fiel, a execução do aplicativo em cada plataforma.

Possivelmente, devido ao seu peculiar funcionamento, o simulador do TotalCross foi o que se mostrou mais rápido para iniciar a execução do aplicativo em desenvolvimento. Além disso, diferentemente dos simuladores presentes nas outras ferramentas, que precisavam ser inicializados previamente, o simulador do TotalCross era inicializado a cada simulação e, mesmo assim, conseguia ser mais veloz.

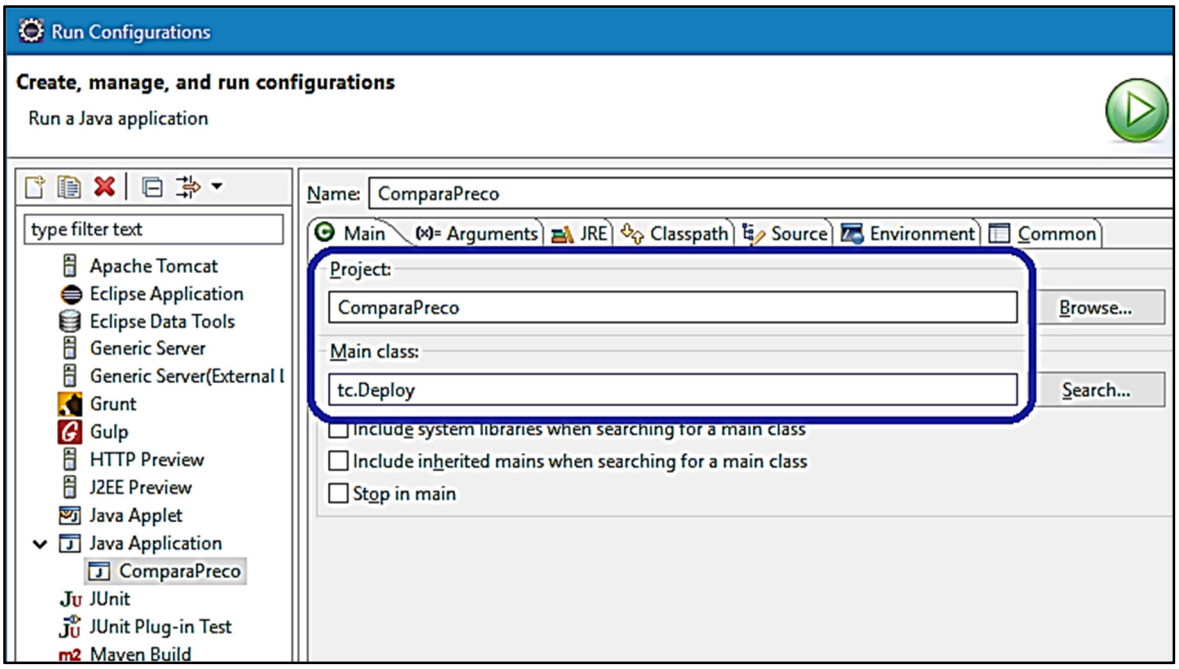
#### 5.3.2.4 *Deploy* e Distribuição

A ferramenta TotalCross possui uma classe responsável pela geração do aplicativo final, ou seja, do arquivo instalador, que será publicado e distribuído na loja de aplicativos da plataforma correspondente. Trata-se da classe `tc.Deploy`, cujo funcionamento é parecido com o da classe `totalcross.Launcher`, visto no uso do simulador.

Durante a execução da classe `tc.Deploy`, foi necessário repassar alguns parâmetros, variando de acordo com a plataforma para a qual estava sendo gerado o aplicativo móvel.

Dentre os parâmetros, estavam a própria plataforma destino, a chave de licenciamento e a classe principal do aplicativo móvel. Na Figura 15 é possível visualizar a definição da classe durante a configuração para iniciar a distribuição do aplicativo.

**Figura 15: Configuração de deploy na IDE Eclipse**




Fonte: Elaborada pelo autor.

Devido as peculiaridades de cada plataforma, principalmente do sistema iOS, no qual a publicação de um aplicativo envolve a utilização de um certificado digital, existe, na documentação da ferramenta, um apoio para executar a parte não automatizada do processo, inclusive a criação do certificado e a geração do arquivo da aplicação já assinado com a chave certificada. Considerando que o processo não pôde ser automatizado, a presença das informações na documentação ajudou consideravelmente na execução do procedimento.

### 5.3.3 Cordova, PhoneGap e Ionic

Devido as enormes similaridades das ferramentas Cordova, PhoneGap e Ionic elas foram agrupadas no desenvolvimento de um único aplicativo. Avaliando a documentação das três não se identificou relevância na utilização de uma ou outra, considerando o desenvolvimento em si, visto que, além de utilizarem as mesmas linguagens e lógica de funcionamento, apoiam-se também nos mesmos *plugins*, na grande maioria fornecida pela Cordova, que permite a utilização ou implementação de recursos específicos. Desta forma, a maior parte das avaliações e considerações foram aplicadas de forma conjunta, comparando

especificamente este tipo de desenvolvimento com os outros dois propostos pelas ferramentas Xamarin e TotalCross.

As ferramentas Cordova, PhoneGap e Ionic compartilham também um peculiar modelo de instalação, baseado no NPM (*Node Package Manager*, traduzido para Gerenciador de Pacotes do Node). O NPM é um repositório *online* de pacotes JavaScript e ao mesmo tempo é um comando que dá acesso a este repositório. O NPM permite fazer o *download*, instalação e configuração das ferramentas, não sendo necessários ajustes adicionais por parte do desenvolvedor. Especificamente, o PhoneGap, além da instalação via NPM, também possui um assistente de instalação em modo gráfico, que pode ser mais amigável, porém, assim como o comando texto, não requer escolhas adicionais por parte do usuário instalador.

Efetivamente, o processo de instalação e configuração foi realizado através do NPM e consistiu simplesmente em executá-lo, aguardando sua conclusão. Não foram registrados problemas ou mesmo dificuldades na execução do procedimento. Os pacotes de instalação baixados e instalados, através do recurso NPM, eram consideravelmente menores entre todas as ferramentas. O pacote de instalação da ferramenta Cordova possui 46,6MB, o do PhoneGap 69,8MB e do Ionic 71,8MB.

As ferramentas Cordova e PhoneGap são gratuitas, com licenciamento *open Source*, conforme Licença da Apache, versão 2.0 de janeiro de 2004, disponível em <http://www.apache.org/licenses/LICENSE-2.0>. Por sua vez, o Ionic possui duas formas de licenciamento, uma é gratuita e, apesar de não restringir funcionalidades disponíveis para o desenvolvedor, possui limitações na quantidade de alguns procedimentos, tais como, quantidade mensal de *deploys* do aplicativo. A forma de licenciamento paga, que custa 20 dólares por mês, possui quantidades limitadas para alguns procedimentos bem maiores que na versão grátis, além de permitir pagar um valor reduzido quando algum destes limites for ultrapassado. Neste trabalho, o Ionic foi utilizado sob o licenciamento gratuito, assim como Cordova e PhoneGap, para os quais não havia outra possibilidade.

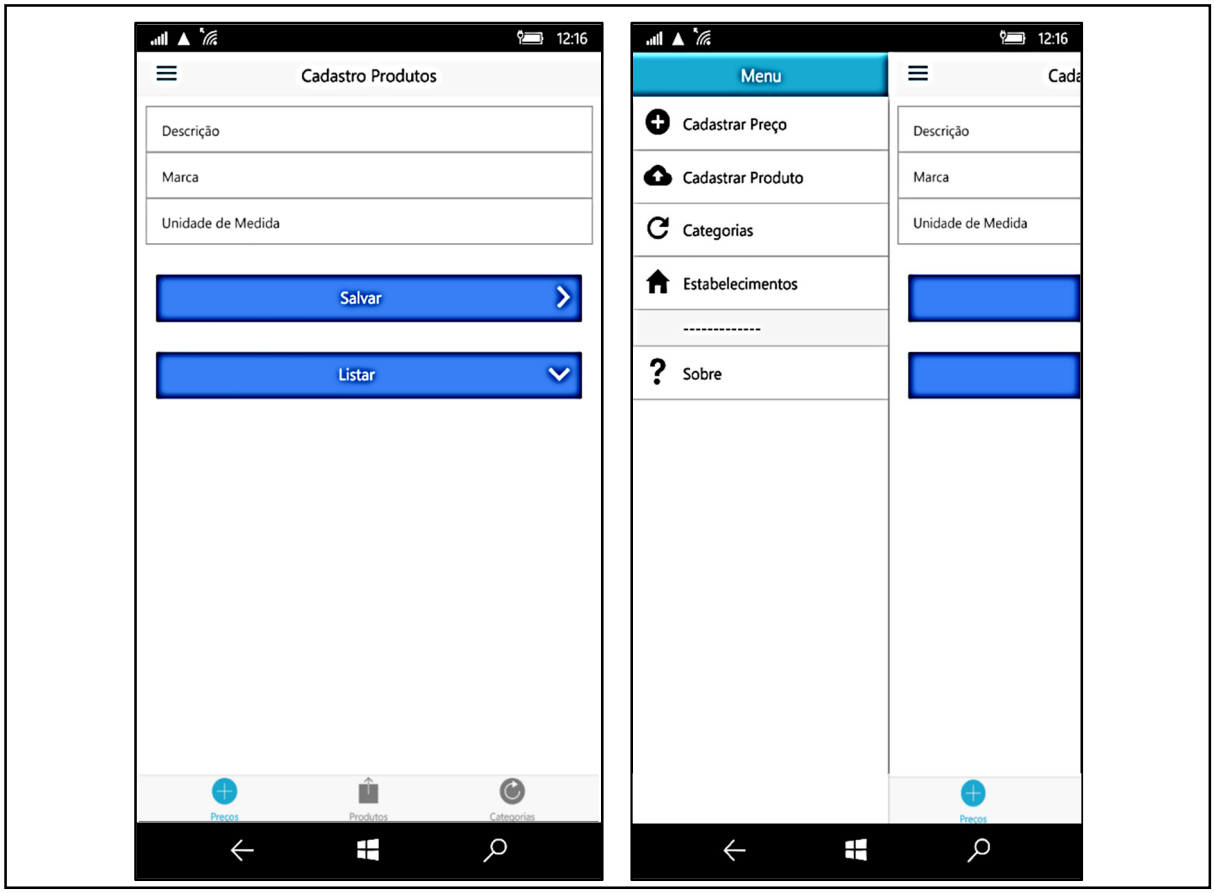
Efetivamente, nenhuma das três ferramentas possui uma IDE nativa, que possa ser executada localmente e utilizada para o desenvolvimento do aplicativo. Devido a este fator, foi utilizado um editor de texto comum, para edição do código-fonte. A ferramenta Ionic, porém, disponibiliza o Ionic *Creator*, que pode ser acessada em um navegador *web* e permitiu o desenvolvimento de parte do aplicativo móvel, mais especificamente do desenho e montagem de parte da interface. Infelizmente, alguns recursos do Ionic *Creator*, especificamente voltados à edição de código-fonte, não puderam ser utilizados na versão gratuita, pois estão disponíveis

apenas ao adquirir o licenciamento pago. Esse fator faz com que seja, praticamente obrigatório, realizar o pagamento da licença mensal para usufruir integralmente da IDE *on-line* e da ferramenta.

Apesar da utilização de HTML, CSS e JavaScript, linguagens bastante conhecidas, o desenvolvimento nestas ferramentas apresentou o inconveniente, que foi, justamente, a falta de uma IDE nativa de qualidade. Obviamente, há no mercado muitos editores e mesmo IDE's que possibilitam a escrita de código, porém, em sua imensa maioria, há poucos recursos de apoio que estariam disponíveis em um componente nativo. Este foi o principal ponto negativo destas ferramentas, perante as demais. Evidentemente, existe a possibilidade de utilizar o Ionic *Creator*, porém, é preciso considerar que a edição de código-fonte só é possível na versão paga da ferramenta. Na Figura 16 é possível visualizar a interface do aplicativo executando no simulador da plataforma Windows Phone e percebe-se que, mesmo sendo moderna e ter uma aparência agradável, ela não está adaptada ao padrão do sistema operacional.

Devido as características das linguagens HTML, CSS e JavaScript, onde a definição e desenho da interface são feitos com as duas primeiras e a execução de rotinas com a terceira, foi possível realizar uma boa separação dos códigos durante o desenvolvimento. Ainda assim, uma única tela exibida para o usuário, na verdade era uma junção de diversos arquivos HTML e CSS que compunham o resultado final. Além disso, as chamadas para as rotinas JavaScript ficaram embutidas no código HTML. Nesse contexto, o desafio foi manter o código enxuto e agregado, de maneira a facilitar manutenções futuras e correções de *bugs*. Na Figura 16 é possível visualizar a interface na plataforma Windows Phone. Para ilustrar melhor a consideração anterior, o menu do aplicativo é formado, na verdade, por um código HTML, isolado em um arquivo único, que depois foi agrupado em uma exibição única para o usuário.

**Figura 16: Aplicativo ComparaPreco executando na plataforma Windows Phone**




Fonte: Elaborada pelo autor.

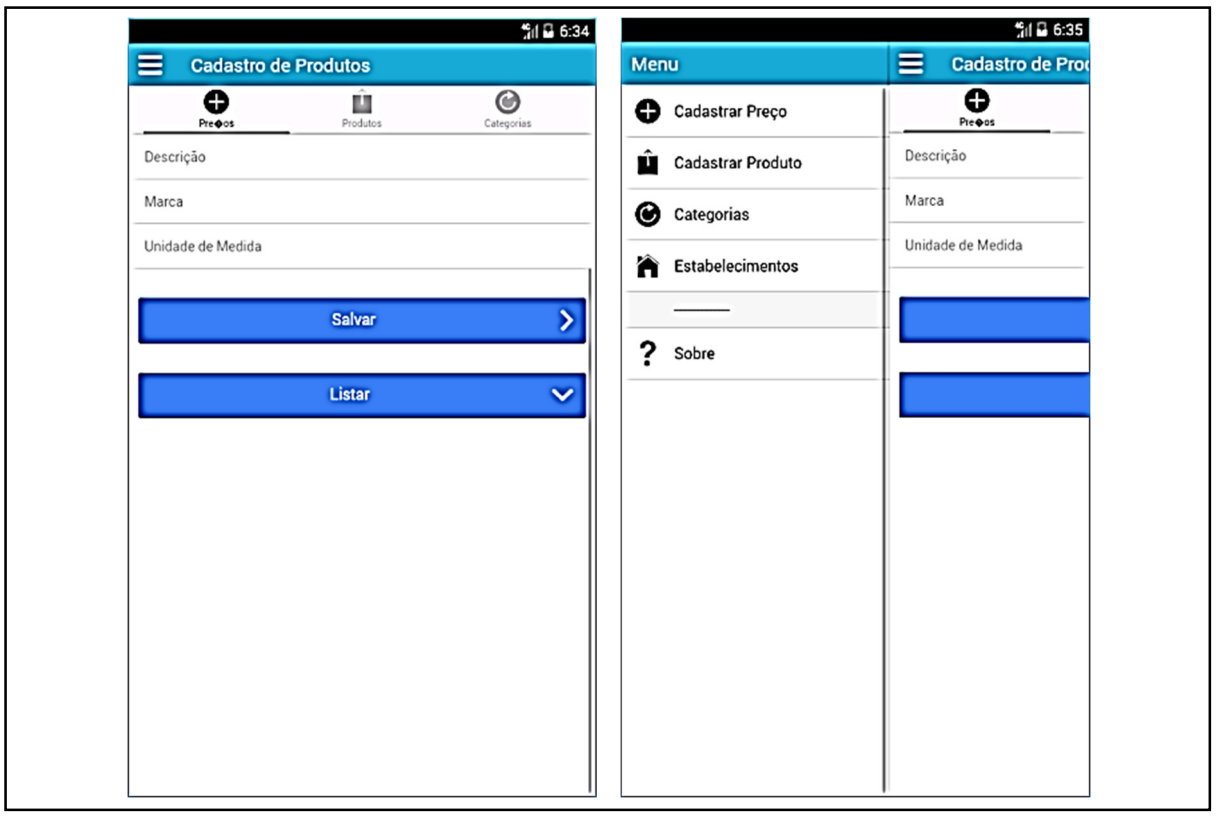
A codificação, assim como ocorreu na TotalCross, foi totalmente multiplataforma, com código unificado, não havendo especificidades para um sistema ou outro. Mesmo quando foram utilizados *plugins* do Cordova para realização de tarefas específicas, foi mantida a escrita de um único código, que durante a execução, gerava o mesmo resultado, independentemente da plataforma onde estava executando.

Pelo fato de ter sido utilizado um editor de texto comum para edição do código-fonte, diversos recursos, normalmente disponíveis em qualquer IDE, não estavam disponíveis. Com isso, foi necessário um cuidado extra, além de um bom conhecimento das linguagens, para identificar com antecedência possíveis erros de escrita, evitando retrabalho e correções de erros em excesso durante a execução de testes.

A interface foi construída com linguagem HTML e CSS e, considerando o potencial já consolidado destas duas linguagens na criação de interface, o visual é excelente e, em alguns casos, o melhor dos três formatos de desenvolvimento, podendo ter superado até mesmo o desenvolvimento nativo, em alguns casos. Ainda assim, não houve muita adequação

automatizada para cada plataforma, apenas pequenos detalhes que, talvez, tenham sido modificados ao executar nos diferentes sistemas devido ao processamento da plataforma e não da ferramenta. Mesmo quando a adequação foi feita manualmente, o resultado foi muito parecido com o apresentado na ferramenta TotalCross, de maneira que a adequação afetou igualmente a execução do aplicativo em todas as plataformas. Na Figura 17, um vislumbre da interface na plataforma Android, exibindo o menu de opções do aplicativo fechado e aberto.

**Figura 17: Aplicativo ComparaPreco rodando na plataforma Android**




Fonte: Elaborada pelo autor.

Objetivando facilitar o processo e, também, testar as ferramentas, o desenho e definição de interface foi realizado através do Ionic *Creator* que, surpreendentemente, possui o recurso de clicar e arrastar de componentes, o que permitiu montar as telas visualmente, sem a necessidade de executar o aplicativo a todo momento em simulador ou emulador para verificar o resultado.

Na Figura 18, pode-se visualizar a interface do aplicativo executando no simulador da plataforma iOS, também com o menu do aplicativo fechado e aberto.

**Figura 18: Aplicativo ComparaPreco executando na plataforma iOS**


Fonte: Elaborada pelo autor.

Foi possível utilizar alguns componentes disponíveis que simulam especificamente uma plataforma ou outra, porém, seu uso fez com que ele fosse apresentado igualmente em todas os sistemas operacionais, não havendo adequação automática. Na Figura 19, é possível visualizar trecho de código HTML, com o qual foi feita a definição de uma tela do aplicativo. Apesar da existência de comandos específicos da ferramenta Ionic, a sintaxe da linguagem HTML é padrão.

**Figura 19: Definição da interface com código HTML**

```
<ion-view view-title="Cadastro de Produtos">
    <ion-content class="padding">
        <div class="list">
            <label class="item item-input">
                <input type="text"
                       placeholder="Descrição">
            </label>
            <label class="item item-input">
                <input type="text"
                       placeholder="Marca">
            </label>
            <label class="item item-input">
                <input type="text"
                       placeholder="Unidade de Medida">
            </label>
        </div>
        <div class="padding">
            <a class="button button-block button-positive icon icon-right ion-
chevron-right" ui-sref="tabs.sobre">Salvar</a>
        </div>
    </ion-content>
</ion-view>
```

Fonte: Elaborada pelo autor.

#### 5.3.3.1 Aprendizado

Certamente, a utilização das linguagens HTML, CSS e JavaScript é um diferencial positivo, já que, estas linguagens possuem uma longa estrada de utilização, além de serem largamente empregadas na construção de *sites* e aplicativos que executam nos navegadores de diferentes dispositivos.

A necessidade de utilizar editores de texto comuns dificultou o aprendizado, pois não estavam disponíveis sugestões ou *auto completes,* relacionados ao contexto das ferramentas, o que obrigava a constante consulta à documentação ou exemplos, que permitissem saber como deviam ser utilizados recursos específicos ou mesmo saber os comandos que estavam disponíveis na ferramenta em uso.

#### 5.3.3.2 Desenvolvimento de funcionalidades específicas

A gravação local de dados foi realizada com o SQLite, disponível nas ferramentas com a utilização do *plugin* `cordova-sqlite-storage`. Com ele foi possível prover facilmente todas as funcionalidades de banco de dados no dispositivo móvel e, ainda, sem preocupações

com a plataforma onde o aplicativo estaria executando. Não apenas os comandos triviais como inclusão ou alteração de dados foi feita com um único código, mas também o acesso ao arquivo do banco de dados. Basicamente, o processo consistiu em inicializar a conexão com o arquivo do banco de dados e, a partir desta conexão, executar as opções necessárias.

Novamente a utilização de um *plugin* da ferramenta Cordova possibilitou a utilização da câmera do dispositivo móvel. O *plugin* utilizado foi o `cordova-plugin-camera` que, além de permitir a captura de imagens a partir da câmera do dispositivo, também poderia ter sido utilizado para acessar as imagens presentes nos álbuns do dispositivo, fazendo com que o usuário pudesse escolher uma imagem gravada previamente.

Não foi necessário tratamento específico para plataforma, sendo isto feito diretamente pelo *plugin*. Seu uso foi extremamente simples e consistiu, basicamente, na definição e inicialização da câmera assim que o aplicativo ou tela do aplicativo era carregada, e deixando a cargo do usuário, através do acionamento de um botão, a captura efetiva da foto.

Estão disponíveis diversos *plugins* para trabalhar com consumo de *web-service*, principalmente, relacionados a leitura de estruturas JSON. Neste trabalho foi utilizada especificamente a `jQuery`, biblioteca específica da linguagem JavaScript, devido a sua simplicidade de utilização, bem como, pelo fato de ser amplamente empregada também em páginas *web* convencionais.

A utilização da biblioteca `jQuery` permitiu, não apenas a unificação de código, deixando-o totalmente independente de plataforma, mas também, sua simplificação, já que, praticamente um único método da biblioteca é capaz de consumir um serviço *web*, obter a resposta JSON e já converter a informações em objetos JSON que podem ser iterados.

O compartilhamento de informações nas ferramentas Cordova, PhoneGap e Ionic foi realizado com o *plugin* `cordova-plugin-socialsharing`. Este *plugin* é totalmente compatível com as plataformas Android, iOS e Windows Phone, além disso, permitiria, não somente o compartilhamento genérico, no qual o usuário do aplicativo escolhe o aplicativo de destino do compartilhamento, mas também o compartilhamento direto em um aplicativo de terceiro específico, tal como, Facebook ou Twitter.

A implementação foi relativamente simples, sendo necessário apenas definir algumas opções do compartilhamento e, então, efetuar a chamada específica de compartilhamento. Não foi necessário código específico por plataforma, deixando assim, o mesmo tratamento e código único para todos os sistemas operacionais.

#### 5.3.3.3 Testes com o aplicativo

As três ferramentas possuem simuladores, que executam em um navegador *web*, e que permitiram testar o aplicativo em desenvolvimento. Os simuladores, diferentemente dos emuladores, apresentaram um comportamento, nem sempre fiel ao dispositivo físico, pois como o nome já indica, correspondem a uma simulação do ambiente onde realmente serão executados.

Apesar disso, o uso dos simuladores apresentou uma grande vantagem em relação aos emuladores. Os tempos de inicialização das simulações sempre foram menores que os tempos de inicialização dos emuladores, utilizados no Xamarin, perdendo apenas para o simulador do TotalCross. A agilidade dos simuladores, neste caso, contribuiu muito na redução de tempo ocioso do programador, que ocorria enquanto aguardava a execução do teste.

#### 5.3.3.4 *Deploy* e distribuição

Foi possível efetuar a liberação do aplicativo para distribuição em qualquer uma das ferramentas, diretamente, executando comandos específicos. Ainda assim, foi preciso efetuar ajustes e configurações adicionais, dependendo da plataforma para qual o aplicativo será compilado.

Felizmente as documentações de cada ferramenta possuem informações de apoio ao desenvolvedor, que foram utilizadas para proceder com as modificações e implementações extras, necessárias para a geração do arquivo de instalação do aplicativo, que poderia ser disponibilizado nas lojas de aplicativos de cada sistema operacional.

# 6 RESULTADOS

Após a análise e comparação das ferramentas feito no desenvolvimento do trabalho, apresentado no Capítulo 5, o Capítulo corrente descreve, especificamente, os resultados encontrados. Para facilitar a explanação dos mesmos, são apresentadas tabelas comparativas e gráficos, que permitem melhor visualização das informações descritas. Além disso, a organização foi baseada na Metodologia apresentada no Capítulo 4.

## 6.1 Instalação e configuração

Apesar de possuir uma instalação extremamente automática, centrada em um assistente, cuja maioria dos passos, consiste apenas em confirmar ou continuar, a ferramenta Xamarin necessitou do maior tempo de instalação e configuração dentre todas. Evidentemente, o *download* dos arquivos, muito maiores que os das outras ferramentas, foi o maior responsável pela demora.

Na Tabela 3 é apresentado um comparativo entre as ferramentas, que destaca os principais quesitos e fatores que impactaram na primeira etapa do desenvolvimento do trabalho, que é justamente, a instalação e configuração das ferramentas.

**Tabela 3: Comparativo do processo de instalação das ferramentas**

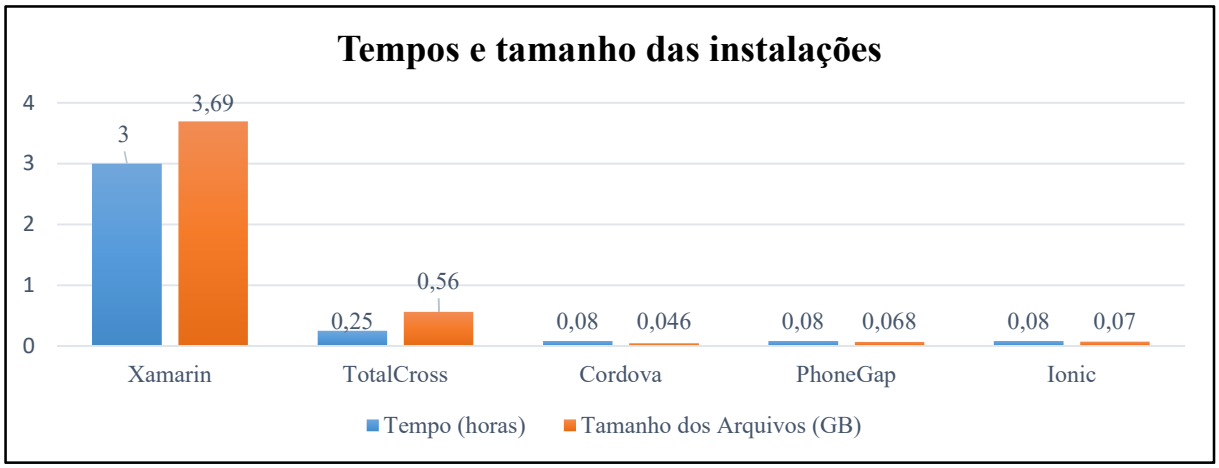
| | **Instalação (com download)** | **Tamanho aproximado** | **Instalação** | **Configuração** |
|---|---|---|---|---|
| Xamarin | ~3 horas | 3,69 Gb – Visual Studio<br>1,67Gb – Xamarin Studio | Automática / Nada complexa | Automática / Nada complexa |
| TotalCross | 15 minutos | 576Mb – Ferramenta e pré-requisitos | Manual com pré-requisitos / Complexidade média | Manual ou com uso de projeto pré-configurado / Complexidade média |
| Cordova | 5 minutos | 46,6Mb – Somente Ferramenta<br>Variável – com *plugins* | Automatizada via NPM / Complexidade baixa | Automática / Baixa complexidade |
| PhoneGap | 5 minutos | 69,8Mb – Somente Ferramenta<br>Variável – com *plugins* | Automatizada via NPM / Complexidade baixa | Automática / Baixa complexidade |
| Ionic | 5 minutos | 71,8Mb – Somente Ferramenta<br>Variável – com *plugins* | Automatizada via NPM / Complexidade baixa | Automática / Baixa complexidade |

Fonte: Elaborada pelo autor.

Claramente, dois aspectos destacaram-se: o tempo de instalação e o tamanho dos

arquivos. Observando a Figura 20, fica ainda mais nítido que o tempo de instalação e configuração está diretamente ligado ao tamanho dos arquivos, de modo que, quanto maior o tamanho dos arquivos, menor é o tempo dedicado ao processo. A relevância do tipo de instalação e configuração, ou seja, de ser manual ou automático é pequena, visto que, não houve grande oscilação de tempo relacionada a este fator.

**Figura 20: Comparativo de tempos de instalação e tamanho dos arquivos**




Fonte: Elaborada pelo autor.

## 6.2 Características

As ferramentas possuem grandes disparidades em relação à suas características. Na Tabela 4 estão destacadas as que possuem maior relevância no que tange ao desenvolvimento de aplicativos multiplataforma. Destaca-se que todas possuem uma forma de licenciamento gratuito, cujas limitações, quando existentes, não influenciaram no desenvolvimento do aplicativo deste trabalho.

**Tabela 4: Principais características das ferramentas**

| | **Licenciamento** | **IDE Nativo / IDE Utilizado** | **Linguagem de programação** | **Definição de Interface (Multiplataforma)** | **Acessa recursos nativos** |
|---|---|---|---|---|---|
| Xamarin | 4 opções: 2 gratuitas e 2 pagas | Sim: Xamarin Studio e Visual Studio com Xamarin | C# | XAML | Sim |
| TotalCross | Grátis | Não: Eclipse | Java | Java Nativa | Sim |
| Cordova | Grátis e de código aberto | Não: Linha de comando e Visual Studio | JavaScript | HTML5 e CSS | Sim |
| PhoneGap | Grátis e de código aberto | Não: Linha de comando | JavaScript | HTML5 e CSS | Não |
| Ionic | 2 opções: 1 grátis e 1 paga | Não: Linha de comando | JavaScript | HTML5 e CSS | Não |

Fonte: Elaborada pelo autor.

### 6.2.1 Modelos de desenvolvimento multiplataforma

Facilmente pode-se distinguir os três modelos de desenvolvimento utilizados nas ferramentas avaliadas. No Xamarin, o aplicativo aparenta ser o mais próximo de um aplicativo nativo, não apenas no desenvolvimento, mas em sua execução. No TotalCross, o conceito do aplicativo executar em uma máquina virtual é levado ao extremo, pois tanto sua implementação como sua execução são isolados do dispositivo/sistema operacional. Já com Cordova, PhoneGap e Ionic, a impressão é estar desenvolvendo uma página *web*, da mesma forma, que ao utilizar o aplicativo, fica evidente também para o usuário esta mesma perspectiva, incluindo até tremulações e pequenas demoras em atualizações ou transições de telas.

Considerando este cenário de diferentes modelos de desenvolvimento multiplataforma, também há outros quesitos de comparação entre as ferramentas, intimamente ligados aos modelos. Eles estão listados na Tabela 5 e representam considerações a se fazer na escolha de uma ou outra ferramenta, visto que nem sempre representam uma deficiência ou diferencial positivo quando avaliados individualmente.

**Tabela 5: Modelos de desenvolvimento**

| **Desenvolvimento** | **Xamarin** | **TotalCross** | **Cordova, PhoneGap, Ionic** |
|---|---|---|---|
| Tipo de Aplicativo | Quase nativo | Encapsulado em máquina virtual | Encapsulado na forma de aplicação *web* |
| Permite acessar recursos nativos | Sim | Parcial | Cordova permite e fornece os recursos para PhoneGap e Ionic |
| Execução do código | Pré-compilado e interpretado no Android e Windows, compilado para ARM no iOS | Interprestado na VM onde está encapsulado | Processado nativamente como página *web* com recursos extras compilados (*plugins* do Cordova) |
| Visual agradável | Sim | Parcial | Sim |
| Transições de tela suaves | Sim | Não | Parcialmente |

Fonte: Elaborada pelo autor.

## 6.3 Código-fonte, aprendizado e tempos de desenvolvimento

O desenvolvimento do código-fonte de um aplicativo é parte essencial, tanto em sua construção, como nas funcionalidades disponíveis para o usuário quando estiver em execução. Evidentemente, o fato de que cada ferramenta emprega diferentes linguagens de programação interfere neste processo. Não que exista alguma relevância direta durante a utilização de um aplicativo móvel, o fato de ter sido utilizada uma ou outra linguagem de programação, porém, durante o desenvolvimento este fator pode ser extremamente importante, tanto na rapidez e

qualidade da implementação, bem como no funcionamento multiplataforma do produto final.

Ao avaliar o desenvolvimento do aplicativo com as diferentes ferramentas, alguns aspectos foram considerados, conforme resultados expressos na Tabela 6. Estes resultados demonstram não apenas grandezas quantitativas, mas também aspectos importantes que impactam diretamente no desenvolvimento do código-fonte.

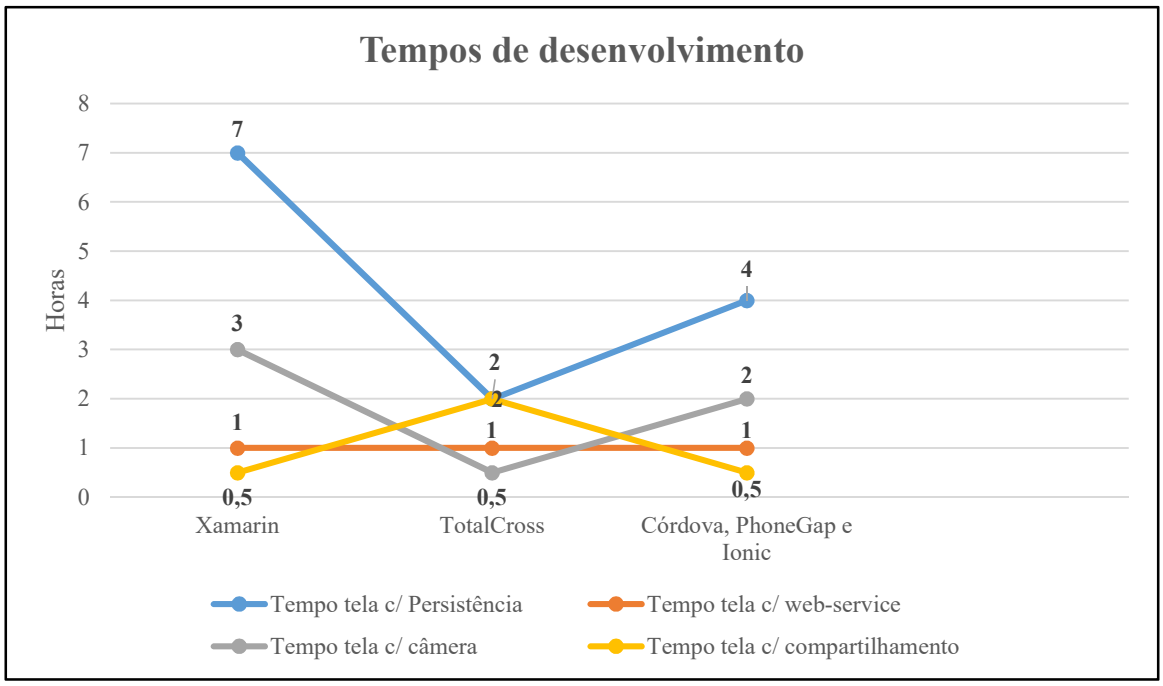
**Tabela 6: Código-fonte: Aprendizado, tempos e pontos de atenção**

| **Código-fonte** | **Xamarin** | **TotalCross** | **Cordova, PhoneGap, Ionic** |
|---|---|---|---|
| Curva de Aprendizagem | Baixa: linguagem conhecida (C#) e boa documentação. Cuidado: Linguagem XAML. | Baixa: linguagem conhecida (Java) e boa documentação. | Média: Falta de IDE e poucos exemplos na documentação. |
| Tempo desenv. de uma tela com persistência de dados local | 7 horas | 2 horas | 4 horas |
| Tempo de desenv. de uma tela com consumo de *web service* | 1 hora | 1 hora | 1 hora |
| Tempo de desenv. de uma tela com acesso à câmera | 3 horas | 0,5 hora | 2 horas |
| Tempo de desenv. de uma tela com compartilhamento de informação | 0,5 hora | 2 horas | 0,5 hora |
| Apoio ao desenvolvedor | IDEs nativas completas, com emulador, *deploy* e publicação na loja de aplicativos integrados. | IDE completa, com simulador e *deploy* integrados. | Facilidade de instalação e documentação. |
| Codificação de cadastro de produtos com 3 campos: código específico por plataforma | 45 linhas: 21% | 0 linhas: 0% | 0 linhas: 0% |
| Codificação de cadastro de produtos com 3 campos: código unificado | 173 linhas: 79% | 202 linhas; 100% | 190 linhas; 100% |
| Ponto mais negativo | Persistência multiplataforma (SQLite) | Mistura de rotinas com desenho de interface | Falta de IDE, fragmentação de códigos |
| Ponto mais positivo | IDE Visual Studio | Persistência multiplataforma (SQLite) | Rotinas simples |

Fonte: Elaborada pelo autor.

Ao observar os itens que podem ser quantificados, é possível melhorar a comparação. Observando a Figura 21, têm-se os seguintes resultados: o desenvolvimento na ferramenta Xamarin foi o mais demorado, a menor oscilação de tempo de desenvolvimento deu-se na ferramenta TotalCross e as ferramentas Cordova, PhoneGap e Ionic apresentaram tempos médios, mas próximos, em padrões vistos no gráfico, da ferramenta Xamarin.

**Figura 21: Comparativo de tempos de desenvolvimento**




Fonte: Elaborada pelo autor.

Ao levar em conta a quantificação de linhas de código específico por plataforma e linhas de código compartilhado faz-se mais uma constatação. Na realidade, quanto mais código específico por plataforma é necessário, mais tempo de desenvolvimento é exigido.

## 6.4 Desenvolvimento da Interface

Assim como o código-fonte, o desenvolvimento da interface é parte fundamental no processo de implementação de um aplicativo móvel. No contexto de aplicação multiplataforma, as ferramentas podem representar um apoio extremamente importante neste quesito.

Um dos itens mais impactantes, sem dúvida, foi a extrema adaptabilidade automática de interface da ferramenta Xamarin. A única exigência para que isso ocorra é a utilização da linguagem XAML, como descritora da interface. Por outro lado, a interface no TotalCross representou a maior frustração, contrastando diretamente com a extrema facilidade com que é implementada, utilizando a própria linguagem Java, seu visual é desagradável, lembrando diretamente aplicações Java *desktop,* extremamente destoante do cenário *mobile*.

A interface, nas ferramentas que utilizam HTML e CSS, fica adequada, ainda que apresente poucas adaptações automáticas para as diferentes plataformas. O Ionic foi uma grata surpresa, pois possibilita o uso do Ionic Creator para montagem da interface com dois incríveis diferenciais, o primeiro é a possibilidade de visualizar a interface em tempo de projeto e, o

segundo, é a possibilidade de escolher os componentes e arrastá-los para a tela, posicionando-os da melhor forma possível.

Incluindo estes pontos de maior relevância, a Tabela 7, exemplifica também, outros quesitos avaliados nas ferramentas testadas, dando um vislumbre mais geral sobre a implementação da interface e os diferenciais, positivos e negativos, de cada ferramenta.

**Tabela 7: Comparativo de recursos para desenvolvimento de interface**

| **Interface** | **Xamarin** | **TotalCross** | **Cordova, Ionic e PhoneGap** |
|---|---|---|---|
| Linguagem para interface multiplataforma | XAML (somente interface) | Java (Interface e rotinas) | HTML e CSS (somente interface) |
| Clicar e arrastar componentes | Não | Não | Parcialmente possível no Ionic |
| Visualização em tempo de projeto | Não | Não | Não |
| Adequação automática multiplataforma | Sim, componente ajusta de acordo com plataforma | Não, componente é sempre o mesmo | Parcial, alguns componentes, como menus e abas são posicionados de acordo com a plataforma. |
| Determinar visualização independente de plataforma | Utilizando XAML é parcial, sendo possível fixar ícones e imagens independente de plataforma. | Sim, visualização é praticamente a mesma sempre. | Parcialmente é possível determinar caraterísticas independentemente da plataforma. |

Fonte: Elaborada pelo autor.

## 6.5 Desenvolvimento de funcionalidades específicas

Sobre a implementação de funcionalidades específicas a consideração mais importante é a extrema restrição na funcionalidade de compartilhamento, possível de desenvolvimento, na ferramenta TotalCross, na qual, foi necessário implementar um envio de e-mail para "compartilhar" informações. Todas as demais ferramentas permitem as funcionalidades, porém, na Xamarin é necessário código específico por plataforma para persistência de dados e manipulação de imagens da câmera.

Na Tabela 8 estão agrupados os quesitos avaliados na implementação de funcionalidades específicas, bem como, as considerações realizadas para cada ferramenta.

**Tabela 8: Implementação de funcionalidades específicas**

| Funcionalidades | Xamarin | TotalCross | Cordova, Ionic e PhoneGap |
|---|---|---|---|
| Persistência local de dados | Permite, com necessidade de código específico por plataforma | Permite, código compartilhado | Permite, código compartilhado |
| Câmera do dispositivo | Permite, com necessidade de código específico por plataforma | Permite, código compartilhado | Permite, código compartilhado |
| Ação "Compartilhar" | Permite, com código compartilhado | Permite com severas restrições, código compartilhado | Permite, código compartilhado |

Fonte: Elaborada pelo autor.

## 6.6 Testes de aplicativos, *deploy* e distribuição

Evidentemente, os primeiros testes de aplicativos móveis são feitos durante o desenvolvimento, com o uso de emuladores ou simuladores, visando agilizar o processo e, reduzir o custo, já que evita a necessidade de compra de equipamentos físicos.

A Tabela 9 destaca a comparação dos quesitos relacionados às etapas de testes, *deploy* e distribuição do aplicativo móvel desenvolvido. Um destaque muito positivo é a possibilidade de integração direta com as lojas de aplicativos, existentes na ferramenta Xamarin, através do Visual Studio.

**Tabela 9: Testes, *Deploy* e distribuição**

| | Xamarin | TotalCross | Cordova, PhoneGap e Ionic |
|---|---|---|---|
| Tamanho do arquivo para distribuição | Android: 8,6Mb<br>Windows Phone: 2,5Mb<br>iOS: 5,3Mb | Android: 7Mb<br>Windows Phone: 1,5Mb<br>iOS: 2,4 Mb | Android: 978Kb<br>Windows Phone: 961Kb<br>iOS: 900Kb |
| Tempo médio de carga do aplicativo no simulador/emulador | Android: 80 seg<br>Windows Phone: 50 seg.<br>iOS: 90 seg | Android: 20 seg<br>Windows Phone: 19 seg<br>iOS: 25 seg | Android: 35 seg<br>Windows Phone: 30 seg.<br>iOS: 40 seg |
| Permite *deploy* de arquivo instalador | Sim, muito simples | Sim, com alguns ajustes manuais | Sim, com muitos ajustes manuais |
| Integração direta com loja de aplicativos | Sim | Não | Não |

Fonte: Elaborada pelo autor.

## 6.7 Avaliação Geral

No contexto geral, todas as ferramentas atingiram o objetivo ao qual se propõem, que é facilitar o desenvolvimento de aplicativos multiplataforma, através do compartilhamento de código-fonte e definição de interface. Ainda assim, durante as implementações realizadas nas ferramentas, sempre foi necessário contornar limitações inerentes as mesmas.

A Tabela 10 apresenta uma comparação abrangente, considerando os principais pontos que foram avaliados, dentre as ferramentas. Estes itens demonstram que a escolha da melhor ferramenta, dependerá muito do olhar crítico do desenvolvedor sobre as funcionalidades e recursos que terá à disposição, bem como das possíveis dificuldades que virá a enfrentar durante o desenvolvimento.

**Tabela 10: Comparativo geral**

| **Item** | **Xamarin** | **TotalCross** | **Cordova, PhoneGap e Ionic** |
|---|---|---|---|
| Produtividade | Baixa | Muito Alta | Alta |
| Codificação | Parcialmente única | Única | Única |
| Interface | Única, adaptação automatica | Única, sem adaptação | Única, pouca adaptação |
| Funcionalidades específicas | Totalmente suportadas | Parcialmente suportadas | Totalmente suportadas |
| Emulação/Simulação | Emulação nativa de cada plataforma | Simulação | Simulação |
| *Deploy* | Parcialmente automatizado | Manual | Manual |
| Manutenção | Complexa | Simples | Regular |

Fonte: Elaborada pelo autor.

Em termos de produtividade, a ferramenta TotalCross demonstrou alto desempenho, permitindo a construção de um aplicativo de forma rápida e confiável. Este resultado provém principalmente pelo fato da ferramenta ter aproveitado todos os recursos da linguagem Java, de modo que, para um desenvolvedor com experiência na mesma, o processo de implementação de um aplicativo móvel será algo trivial. O Xamarin, por sua vez, mostrou-se menos produtivo. Considerando sua proposta de produção de aplicativos bastante fiéis aos fabricados de forma nativa, isto é justificável, afinal, o resultado realmente surpreende no quesito fidelidade à cada plataforma. Outro fator que contribuiu para a menor produtividade foi a necessidade de codificação específica para as plataformas. As ferramentas Cordova, PhoneGap e Ionic são produtivas, principalmente devido à utilização das linguagens HTML, CSS e JavaScript, que além de conhecidas, possuem uma sintaxe conhecida e de fácil aprendizado.

Sobre a codificação unificada, o Xamarin não se saiu bem. Sempre que foi necessário desenvolver algum recurso fora do extremamente comum, houve a obrigatoriedade de escrever código específico. Este fator impacta tanto na produtividade, como na manutenção futura do código, já que o desenvolvedor precisará tomar cuidado para fazer todos os ajustes necessários, também na codificação específica. As demais ferramentas permitiram o compartilhamento total de todo o código, facilitando não só o desenvolvimento inicial, mas também a necessidade de futuras modificações.

A interface automaticamente ajustável para cada plataforma no Xamarin representou, dentre as três ferramentas, o melhor desempenho neste quesito, pois além de permitir uma única definição, na qual apenas ajustes relativos a localização dos ícones foi necessária, a exibição das telas em cada sistema operacional ficou extremamente fiel a interface de aplicativos nativamente desenvolvidos. Por outro lado, a interface implementada na ferramenta TotalCross foi a mais prejudicada, pois praticamente permaneceu sempre a mesma, independentemente da plataforma onde era mostrada. Já nas ferramentas Cordova, PhoneGap e Ionic a interface também foi definida de forma única e pouquíssimos ajustes automáticos foram realizados pelas ferramentas.

O desenvolvimento de funcionalidades específicas apenas não foi totalmente possível na ferramenta TotalCross, onde o recurso de compartilhamento de informações foi implementado, porém, ficou limitado ao envio de e-mail diretamente do aplicativo, não permitindo, nem mesmo, o acionamento do aplicativo de e-mail padrão do sistema operacional. Nas demais ferramentas a funcionalidade de compartilhamento apresentou o comportamento esperado. As funcionalidades de gravação em banco de dados local e acesso à câmera do dispositivo foram possíveis em todas as ferramentas, ainda que, exigiram, na ferramenta Xamarin, a divisão do código-fonte em compartilhado e específico por plataforma.

A realização de testes do aplicativo durante o desenvolvimento, feita com o apoio dos emuladores e simuladores, pode ser dividida em dois cenários: na ferramenta Xamarin, a utilização dos emuladores nativos de cada plataforma permite uma execução do aplicativo muito mais realista e próxima do funcionamento em um dispositivo físico, porém, é importante mencionar que a execução no emulador é consideravelmente mais demorada, já que o emulador trata-se, na verdade, de uma máquina virtual que executa a plataforma e sistema operacional correspondente. Por outro lado, nas ferramentas TotalCross, Cordova, PhoneGap e Ionic os testes foram executados em simuladores, cujo desempenho é muito superior, permitindo uma rápida inicialização, tanto do simulador em si, como do teste do aplicativo. Evidentemente, pelas características do simulador, que não retrata fielmente o ambiente real de um dispositivo, a execução do aplicativo pode não corresponder fielmente ao funcionamento no equipamento físico.

Ao final do desenvolvimento a geração do pacote de instalação do aplicativo, para disponibilização na loja de aplicativos de cada plataforma pode ser feita em todas as ferramentas, ainda que, o Xamarin possua recursos mais avançados neste quesito, tanto em relação aos ajustes necessários para cada plataforma, bem como a geração do arquivo em si e,

mesmo, sua transferência para a loja de aplicativos do sistema operacional correspondente. As documentações de todas as ferramentas possuem orientações sobre o processo de geração do aplicativo, mas, na ferramenta Ionic elas são extremamente simples e não permitiriam o conhecimento pleno exigido para a tarefa, obrigando consultas adicionais em outras publicações ou mesmo nas plataformas destino do aplicativo.

Após a liberação do aplicativo, pode surgir eventualmente, a necessidade de realizar manutenções, tanto para correção de *bugs* como para o incremento de novas funcionalidades. As ferramentas TotalCross, Cordova, PhoneGap e Ionic mostraram-se mais favoráveis, talvez pelo fato de possuírem uma estruturação mais simples em relação aos *plugins* e bibliotecas externas. Já na ferramenta Xamarin, mesmo durante o desenvolvimento, atualizações em bibliotecas externas e do próprio Xamarin, geraram inconvenientes, sendo necessário, por muitas vezes, a reinstalação de pacotes externos no projeto do aplicativo para que o funcionamento voltasse ao normal e não exibisse erros que não existiam. Por sinal, nestas situações os erros indicados, além de não existirem, não eram claros, impedindo que fossem feitos ajustes manuais para corrigir os problemas. O ambiente do Visual Studio e do próprio Xamarin são complexos, de modo que, mesmo com todos os ajustes automatizados ainda carecem de melhorias para comportar atualizações de componentes internos de modo que não interfiram no bom funcionamento dos aplicativos em desenvolvimento.

# 7 CONCLUSÃO E TRABALHOS FUTUROS

Em um cenário extremamente competitivo e versátil a adaptação rápida às oscilações locais e globais do mercado torna-se um diferencial para os desenvolvedores de aplicativos. Não menos importante é a abrangência de seu produto em relação ao público, garantindo o acesso padronizado a todos, de maneira oficial, independentemente da plataforma que utilizam. Além disso, o fato de não focar, especificamente apenas em um ou outro sistema operacional móvel como base de desenvolvimento, minimiza os riscos relacionados a modificações abruptas em sua adoção.

Do ponto de vista do usuário, a liberdade na escolha da plataforma e sistema operacional é garantida, tendo em vista que o desenvolvimento multiplataforma permite acesso aos aplicativos de maneira democrática, o fator de escolha relacionado à quantidade de aplicativos disponíveis na loja de cada sistema móvel torna-se menos relevante, enfatizando assim, outros quesitos a serem considerados na escolha. Importante considerar também que, a existência de mais possibilidades para o consumidor tende a evitar a monopolização do mercado e estagnação do progresso contínuo, seja no acréscimo de novas funcionalidades ou de melhorias e correções de *bugs*.

O desenvolvimento multiplatafoma é uma realidade, fomentado tanto pelas empresas, cujos produtos são ferramentas de desenvolvimento para este fim, como também, pelo mercado, onde há uma saudável competição entre fabricantes e sistemas operacionais móveis. A necessidade de criar soluções de maneira rápida e com qualidade, conseguindo atingir o maior público possível, faz com que haja um esforço coletivo na adoção e popularização de recursos que apoiem e facilitem este trabalho. Indiscutivelmente, as ferramentas testadas servem muito bem a este propósito e sua utilização traz benefícios aos desenvolvedores, consumidores e usuários de aplicativos móveis.

As ferramentas avaliadas apresentaram um bom desempenho geral e representam uma excelente alternativa ao desenvolvimento nativo de aplicativos móveis. Com elas é possível desenvolver tranquilamente para as três plataformas móveis de maior expressão mundial, com a possibilidade de implementar praticamente todas as funcionalidades nativas. Ainda que existam limitações variadas, a escolha de uma outra ferramenta deve considerar também benefícios e o foco de desenvolvimento desejado.

Ao considerar a necessidade de uma aparência próxima a de aplicativos nativos a escolha mais correta é a ferramenta Xamarin, pois, além do aspecto visual extremamente

semelhante a cada sistema operacional, este ajuste é praticamente automatizado pela ferramenta, minimizando o esforço do desenvolvedor no ajuste de peculiaridades da interface para cada plataforma móvel.

As ferramentas Cordova, PhoneGap e Ionic também produzem aplicativos de aparência agradável, porém, com um grau mínimo de ajuste para as diferentes plataformas, deixando a execução mais padronizada em todos os sistemas. Se o objetivo é criar um aplicativo baseado em um *site* da internet, com certeza, estas ferramentas seriam uma escolha muito adequada, pois manteriam o padrão do endereço *web*, porém no formato de um aplicativo.

Quando o foco de desenvolvimento é o meio corporativo, o TotalCross mostra-se como uma excelente opção. Esta constatação surge, principalmente, ao considerar a produtividade atingida com a ferramenta. Apesar de gerar uma interface de menor impacto visual, cadastros e aplicativos voltados a negócios podem ser produzidos de maneira rápida e precisa. A agilidade é provida também pelo simulador próprio, no qual são executados os testes do aplicativo em desenvolvimento.

Não foi possível determinar uma ferramenta que sobrepujasse totalmente as demais. Todas possuem características positivas e negativas que podem variar perante o cenário de desenvolvimento, que depende de diversos fatores, desde o meio em que o aplicativo móvel será utilizado, assim como as funcionalidades mínimas requeridas pelo mesmo, além de custos e prazos de desenvolvimento envolvidos. Obviamente, o conhecimento das limitações e recursos disponíveis nas ferramentas garante que a escolha seja feita de maneira sólida e consistente.

Evidentemente, pelas limitações de tempo e escopo do trabalho, não foram avaliados todos os quesitos e funcionalidades possíveis de um aplicativo móvel, sendo assim, como proposta de trabalhos futuros está a realização de comparativos de desenvolvimento multiplataforma relacionados a pontos ou objetivos específicos como, desenvolvimento de jogos, utilização de sensores, tais como GPS, Acelerômetro, Giroscópio, entre outros disponíveis. Além disso, uma possibilidade de trabalho futuro também é a avaliação do uso de ferramentas de desenvolvimento multiplataforma no ensino da disciplina Programação para dispositivos móveis da UNISC, validando se é possível sua utilização também como ferramenta de ensino.

# 8 REFERÊNCIAS

AMBROS, Luisa. Diferença entre Aplicativos Nativos, Híbridos e Mobile Web Apps. Set-2013. Disponível em: <http://www.luisaambros.com/blog/diferenca-entre-aplicativos-nativos-hibridos-e-mobile-web-apps/ >. Acesso em 04 jun. 2016.

APPLE. Xcode Documentation. 2016. Disponível em: <https://itunes.apple.com/br/app/xcode/id497799835?ls=1&mt=12>. Acesso em 31 mai. 2016.

BRIGGS, Asa; BURKE, Peter. Uma história social da mídia: De Gutenberg à internet. 3. ed. Rio de Janeiro: Zahar, 2016.

BUDIU, Raluca. Mobile: Native Apps, Web Apps, and Hybrid Apps. Nielsen Norman Group, Set. 2013. Disponível em: <https://www.nngroup.com/articles/mobile-native-apps/>. Acesso em 04 jun. 2016.

CAFÉ, Adriel Almeida. Desenvolvimento de Cross-Platform Mobile Apps Utilizando o Titanium Mobile. 2012. 95 p. TCC (Graduação) Faculdade Zacarias de Góes (FAZAG) – Valença, 2012.

CORDOVA. Documentation. 2015. Disponível em: <https://cordova.apache.org/docs/en/latest/guide/overview/>. Acesso em 23 mar. 2016.

DEITEL, H.M.; DEITEL, P.J. Java, como programar. Traduzido por Carlos Arthur Lang Lisbôa. 4. ed. Porto Alegre: Bookman, 2003.

DEITEL, H.M.; DEITEL, P.J.; LISTFIELD J.; NIETO, T.R.; YAEGER C.; ZLATKINA M. C# - Como programar. Traduzido por João Eduardo Nóbrega Tortello e revisão técnica Alvaro Antunes. São Paulo: Pearson Makron Books, 2003.

FAYAD E, M.; SCHMIDT D., C. Object-oriented Application frameworks. Communications of the ACM, volume 40, Issue 10. pág. 32-38. New York: ACM, 1997.

FONSECA, Marcos Roberto da; BEDER, Delano Medeiros. Aplicativos Android: desenvolvimento nativo x uso de ferramentas baseadas em padrões web. Revista T.I.S. São Carlos, v. 4, n. 1 , p. 78-87, jan-abr 2015.

GOOGLE. Android Studio Documentation. 2016. Disponível em: <https://developer.android.com/studio/intro/index.html>. Acesso em 25 mai. 2016.

HEIDENHEIMER, A. J.; HECLO, H., ADAMS, C. T. Comparative Public Policy: The Politics of Social Choice in America, Europe, and Japan. 3.ed. St. Martin's Press, 1983.

IDC. Worldwide Smartphone Market Will See the First Single-Digit Growth Year on Record, According to IDC. IDC: Framingham, 2015. Disponível em <http://www.idc.com/getdoc.jsp?containerId=prUS40664915>. Acesso em 06 mar. 2016.

IONIC. Ionic Documentation Overview: Documentation, 2016. Disponível em: <http://ionicframework.com/docs/overview/>. Acesso em 22 mar. 2016.

LECHETA, Ricardo. Desenvolvendo para iPhone e iPad: Aprenda a desenvolver aplicativos utilizando iOS SDK. 4. ed. São Paulo: Novatec Editora, 2016.

LEVY, Eduardo. Apresentação feita em 24/11/2015 sobre o Setor de Telecomunicações em 2015 em Brasília. Telebrasil: Brasília, 2015. Disponível em: <http://www.telebrasil.org.br/component/docman/doc_download/1520-24-11-215-setor-de-telecomunicacoes-em-2015?Itemid=>. Acesso em 07 mar. 2016.

MEYER, Maximiliano. A história do Android. Oficina da Net, mai-2015. Disponível em: <https://www.oficinadanet.com.br/post/13939-a-historia-do-android>. Acesso em 15 mai. 2016.

MICROSOFT. Visual Studio 2015 Documentação. 2016. Disponível em: <https://www.visualstudio.com/pt-br/products/vs-2015-product-editions.aspx>. Acesso em 17 mai. 2016.

_____. Visual Studio Tools for Apache Cordova Documentation, 2016. Disponível em: <https://taco.visualstudio.com/en-us/docs/>. Acesso em 24 mai. 2016.

_____. Xamarin e Visual Studio Documentação. 2016. Disponível em: <https://msdn.microsoft.com/pt-br/library/mt299001.aspx>. Acesso em 04 jun. 2016.

MOBILETIME. Uso dos aplicativos móveis cresceu 58% em 2015. Mobiletime: São Paulo, 2016. Disponível em: <http://www.mobiletime.com.br/05/01/2016/pesquisa-uso-dos-aplicativos-moveis-cresceu-58-em-2015/425202/news.aspx?noticiario=TT>. Acesso em 07 mar. 2016.

MÔNACO, Thiago; CARMO, Rodolpho Marques do. Desenvolvimento aplicações para Windows Phone. Rio de Janeiro: Brasport, 2012.

MONO. Mono Project Documentation. 2016. Disponível em: <http://www.mono-project.com/docs>. Acesso em 01 jun. 2016.

MORIMOTO, Carlos E. Smartphones, Guia Prático. São Paulo: Guia do Hardware.net, 2009. Disponível em: <http://www.hardware.com.br/livros/smartphones/>. Acesso em 05 mai. 2016.

PETZOLD, Charles. Creating Mobile Apps with Xamarin.Forms: cross-platform C# programming for iOS, Android, and Windows Phone. 2.ed. 2015. Disponível em: <http://download.microsoft.com/download/7/4/7/747DDA87-CA9F-4394-9B3F-4426BFE52BBF/Microsoft_Press_eBook_Xamarin_Preview_2_PDF.pdf>. Acesso em 11 mar. 2016.

PHONEGAP. Desenvolvido por Adobe Systems Inc. 2016. Apresenta informações gerais sobre a ferramenta. Disponível em <http://www.phonegap.com/>. Acesso em: 23 mar. 2016.

PREZOTTO, Ezequiel Douglas; BONIATI, Bruno Batista. Estudo de Frameworks Multiplataforma Para Desenvolvimento de Aplicações Mobile Híbridas. Anais do EATI, ano 4. n. 1, p. 72-79. Universidade Federal de Santa Maria (UFSM). nov. 2014.

SHIRKY, Clay. Lá vem todo mundo: O poder de organizar sem organizações. Rio de Janeiro: Zahar, 2012.

SILVA, André. Aplicações Mobile com o Xamarin Studio. Revista Programar, Coimbra, n.51, dez-2015. Disponível em: <http://www.revista-programar.info/artigos/aplicacoes-mobile-com-o-xamarin-studio/>. Acesso em 09 mar. 2016.

SOUZA, Bruno; NARDON, Fabiane Biznella; REHEM, Serge. A História da Tecnologia Java. Revista Easy Java Magazine 1. mai-2010. Disponível em: < http://www.devmedia.com.br/a-historia-da-tecnologia-java-easy-java-magazine-1/18446>. Acesso em 20 mar. 2016.

SOUZA, Édipo da Silva. Uma análise comparativa de ferramentas de desenvolvimento multiplataforma para dispositivos móveis. 2014. 46 p. Monografia (Graduação em Sistemas e Mídias Digitais) Instituto UFC Virtual da Universidade Federal do Ceará – Fortaleza, 2014.

THOMAZ, Leonardo. Windows em dispositivos móveis – Uma história de design. WinCentral, 2015. Disponível em: <http://www.wincentral.com.br/2015/05/windows-em-dispositivos-moveis-uma-historia-de-design/>. Acesso em 08 mai. 2016.

TOTALCROSS. Documentation. 2016. Disponível em: <http://www.totalcross.com/documentation/companion.html>. Acesso em 20 mar. 2016.

_____. Introdução ao TotalCross. out-2015. Disponível em: <http://www.lab27.com.br/introducao-ao-totalcross>. Acesso em 20 mar. 2016.

W3C. HTML5: A vocabulary and associated APIs for HTML and XHTML - W3C Recommendation 28 October 2014. Out-2014. Disponível em <https://www.w3.org/TR/html5/>. Acesso em 22 mar. 2016.

ZUCKER, Daniel; RISCHPATER, Ray. Beginning Nokia Apps Development: Qt and HTML5 for Symbian and MeeGo. New York: Apress, 2011.