UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Pedro Danilo Ferreira Veloso

ANÁLISE COMPARATIVA NO DESENVOLVIMENTO DE APLICATIVOS UTILIZANDO OBJECTIVE-C E SWIFT

Recife

2019

Pedro Danilo Ferreira Veloso

ANÁLISE COMPARATIVA NO DESENVOLVIMENTO DE APLICATIVOS UTILIZANDO OBJECTIVE-C E SWIFT

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Wilson de Oliveira

Recife

2019

A Deus. À Universidade.

Agradecimentos

Agradeço a Deus, que fez tudo isso possível. À Universidade que me ajudou a trilhar o meu caminho. Aos familiares e amigos que nunca me deixaram desistir. Aos professores que me acompanharam nesta imensa jornada. Às pessoas que chegaram e já foram... e às que chegaram e permaneceram em minha vida. À todas as pessoas que, de uma forma ou de outra, me fizeram ser quem sou hoje. Muito obrigado, pessoal.

“Todo o que o Pai me der virá a mim, e quem vier a mim eu jamais rejeitarei.”
(Jesus Cristo - João 6:37)

Resumo

Vemos as aplicações móveis como um fenômeno de grande significado técnico, social e econômico. Empresas a todo momento tentam tirar algum proveito das tecnologias, sobretudo as móveis, por entenderem que os usuários, nos dias atuais, possuem uma ferramenta de alto poder de desempenho na palma da mão. Empresas como Google e Apple, detém os sistemas operacionais móveis mais utilizados da atualidade: Android e iOS, respectivamente. Atualmente, quando nos referimos a desenvolvimento nativo de aplicações iOS, duas linguagens de programação são bastante usadas, são elas Objective-C e Swift. A primeira delas foi, durante muito tempo, a principal linguagem de programação utilizada para escrever aplicações para OS X e iOS. Trata-se de um superconjunto da linguagem de programação C, e fornece recursos orientados a objetos e tempo de execução dinâmico. Já o Swift foi projetado pela própria Apple em 2010 visando substituir em médio/longo prazo a linguagem Objective-C. Com a promessa de ser mais rápida e possuir uma sintaxe mais simples, ganhou boa parte dos desenvolvedores iOS. O Swift foi incluído no Xcode desde a versão 6. Este estudo tem como objetivo apresentar os conceitos fornecidos nas linguagens Objective-C e Swift, bem como uma análise comparativa dos aspectos de codificação (quantidade de arquivos e caracteres por código-fonte), performance (utilização de CPU, memória e tempo de inicialização) e informações de armazenamento em disco (tamanho do pacote de instalação, e do aplicativo instalado) entre estas duas linguagens. A Ferramenta utilizada para fazer as avaliações é provida pela própria IDE XCode, chamada XCode Instruments. Para realizar a análise foi desenvolvido um aplicativo utilizando as duas linguagens, e em seguida foram feitas rodadas de teste para cada funcionalidade. Ao final foi avaliado que a linguagem Objective-C apresentou uma vantagem de aproximadamente 10.5% quanto a utilização memória, de aproximadamente 98.3% quanto ao armazenamento em disco do arquivo *.xcarchive*, de aproximadamente 94.4% quanto ao tamanho do aplicativo instalado e de aproximadamente 63.7% quanto ao tempo de inicialização do aplicativo em comparação ao Swift. O Swift se mostrou vantajoso quanto a aspectos de codificação, sendo aproximadamente 24.2% menor que o Objective-C em relação a quantidade de código e 31.4% em relação a quantidade de arquivos.

Palavras-chave: iOS, Objective-C, Swift, Apple, aplicativos.

Abstract

We see mobile applications as a phenomenon with significant technical, social and economic significance. At all times, companies try to take advantage of the technologies, especially mobile ones, because they understand that users, nowadays, have a high power performance tool in their hands. Companies like Google and Apple own the most used mobile operating systems: Android and iOS, respectively. Currently, when we refer to the native development of iOS applications, two programming languages come to the mind: Objective-C and Swift. The former was, for a long time, the principal programming language used to write applications for OSX and iOS. Objective-C is a superset of the C programming language and provides object-oriented and dynamic run-time capabilities. Apple designed Swift in 2010, aiming to replace Objective-C in a medium/long term. With the promise of being faster and having simpler syntax, most of the iOS developers adopted Swift. Swift is included in XCode since version 6. This study proposes to approach concepts related to Objective-C and Swift, as well as a comparative analysis of the aspects of coding (number of files and characters by source code), Performance (CPU usage, memory and initialization time) and disk storage information (installation package size, and installed application) between these two languages. The tool used to make the XCode IDE provides the assessments, called XCode Instruments. The analysis was performed by developing an application using the two languages and then were made test rounds for each functionality. In the end it was estimated that the Objective-C language had an advantage of approximately 10.5% related to memory, approximately 98.3% in disk storage of the .xcarchive file, approximately 94.4% in size of installed application and approximately 63.7% of application startup time compared to Swift. Swift proved to be advantageous in coding aspects, being approximately 24.2% smaller than Objective-C for code quantity and 31.4% for file quantity.

Keywords: iOS, Objective-C, Swift, Apple, Applications.

Lista de ilustrações

Figura 1 – Arquitetura do sistema operacional iOS 15
Figura 2 – Estrutura do arquivo JSON 21
Figura 3 – Tela inicial do aplicativo 22
Figura 4 – Tela de resultados 22
Figura 5 – Tela de mais sorteados 23
Figura 6 – Fórmula para cálculo da probabilidade 23
Figura 7 – Tela de probabilidades 24
Figura 8 – Debug Navigation 26
Figura 9 – Memory Report 27
Figura 10 – Representação gráfica do uso médio de CPU por funcionalidade . . 29
Figura 11 – Representação gráfica do uso médio de memória por funcionalidade 30
Figura 12 – Representação gráfica do armazenamento do arquivo *xcarchive* e pós instalação 31
Figura 13 – *Output* do tempo de inicialização. 32
Figura 14 – Representação gráfica da quantidade de arquivos 32

Lista de tabelas

Tabela 1 – Representação dos dados de uso médio de CPU 29
Tabela 2 – Uso médio de Memória (valores em *megabytes*) 30
Tabela 3 – Armazenamento - Arquivo *xcarchive* 31
Tabela 4 – Armazenamento - Aplicativo instalado 31
Tabela 5 – Tempo de inicialização em *ms* 32
Tabela 6 – Quantidade de caracteres 33
Tabela 7 – Teste de tempo de inicialização em milissegundos 36
Tabela 8 – Médias percentuais do teste utilização de CPU - Swift 37
Tabela 9 – Médias percentuais do teste utilização de CPU - Objective-C 38
Tabela 10 – Teste utilização de Memória - Swift (valores em *megabytes*) 39
Tabela 11 – Teste utilização de Memória - Objective-C (valores em *megabytes*) . 40
Tabela 12 – Arquivos do aplicativo Objective-C e Swift 41

Lista de abreviaturas e siglas

IDE	Integrated Development Environment
IPA	iOS Package Application
SDK	Software Development Kit
REPL	Read–Eval–Print Loop
CPU	Central Processor Unit
UI	User Interface
OS	Operating System

Sumário

Lista de ilustrações 7

1 INTRODUÇÃO 12
1.1 Motivação e Justificativa 12
1.2 Objetivos 13
1.3 Artefatos 13
1.4 Estrutura do trabalho 14

2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 Sistema operacional iOS 15
2.1.1 AppStore e Desenvolvimento de Aplicativos 16
2.2 Objective-C 16
2.3 Swift 17
2.4 Trabalhos relacionados 18

3 METODOLOGIAS 20
3.1 Artefato e Caso de Uso 20
3.2 Ambiente de desenvolvimento 24
3.3 Metodologia de avaliação 25
3.3.1 Ferramenta para extração dos dados 26

4 ANALISE COMPARATIVA 28
4.1 Resultados 28
4.1.1 Análise de CPU 28
4.1.2 Uso de memória 29
4.1.3 Armazenamento 30
4.1.4 Tempo de inicialização 31
4.1.5 Quantidade de arquivos 32
4.1.6 Quantidade de código 33

5 CONCLUSÃO E TRABALHOS FUTUROS 34
5.1 Dificuldades encontradas 34
5.1.1 Trabalhos futuros 35

6 APÊNDICE 36
6.1 Tabelas referentes aos testes e comparações 36

REFERÊNCIAS 42

1 Introdução

Vemos as aplicações móveis como um fenômeno de grande significado técnico, social e econômico. Empresas a todo momento tentam tirar algum proveito das tecnologias, sobretudo as móveis, por entenderem que os usuários, nos dias atuais, possuem uma ferramenta de alto poder de desempenho na palma da mão. Empresas como Google e Apple, detém os sistemas operacionais móveis mais utilizados da atualidade: Android e iOS, respectivamente.

Atualmente, quando nos referimos a desenvolvimento nativo de aplicações iOS, duas linguagens de programação são bastante lembradas, são elas Objective-C e Swift. A primeira delas foi, durante muito tempo, a principal linguagem de programação utilizada para escrever aplicações para OS X e iOS. Trata-se de um superconjunto da linguagem de programação C, e fornece recursos orientados a objetos e tempo de execução dinâmico. Já o Swift foi projetado pela própria Apple em 2010 visando substituir em médio/longo prazo a linguagem Objective-C. Com a promessa de ser mais rápida e possuir uma sintaxe mais simples, ganhou boa parte dos desenvolvedores iOS. O Swift foi incluído no Xcode desde a versão 6.

Nesse contexto, este estudo tem como objetivo a abordagem de conceitos relacionados a Objective-C e Swift, bem como uma análise comparativa dos aspectos de codificação (sintaxe, disponibilidade de materiais e códigos fontes), performance (utilização de CPU, memória e tempo de inicialização) e informações de armazenamento em disco (tamanho do pacote de instalação, e do aplicativo instalado) entre estas duas linguagens. A Ferramenta utilizada para fazer as medições é provida pela própria IDE XCode, chamada *Debug Navigator* que possui integração com outra poderosa ferramenta chamada XCode *Instruments*.

1.1 Motivação e Justificativa

Conhecido como um dos melhores sistemas operacionais do mundo, o iOS é famoso por sua segurança, estabilidade e confiança. Atualmente o iOS é o segundo mais utilizado no mundo, perdendo para o Android (HOLST, 2019), um possível motivo seria a liberdade e a compatibilidade com equipamentos de diversas marcas, oferecida pelo sistema concorrente. A Apple, por sua vez, mantém total controle sobre o seu sistema. Aplicativos são desenvolvidos e disponibilizados todos os dias e segundo dados do início de 2017 haviam 2,2 milhões de aplicativos na App Store (GROTHAUS, 2018). A maioria dessas aplicações foram desenvolvidas em Swift ou Objective-C, sendo a primeira delas a linguagem mais popular no desenvolvimento nativo de aplicativos para

iOS atualmente.

O presente estudo tem como motivação analisar e comparar as duas principais linguagens de programação disponíveis no que diz respeito à desenvolvimento de aplicações iOS, afim de contribuir na comparação entre as duas linguagens A análise comparativa também visa fornecer dados para auxiliar desenvolvedores e profissionais de tecnologia em processos de tomada de decisão sobre qual linguagem utilizar no desenvolvimento de aplicações iOS, considerando a necessidade de processar e exibir grandes quantidades de dados, inclusive em tempo real. Mercados como o financeiro e o de serviços de *streaming*, que estão cada vez mais comuns, precisam operar com o menor atraso possível pois, em alguns casos como no mercado financeiro de ações, um atraso por conta de mau desempenho pode significar perdas significativas de dinheiro.

Neste contexto o presente estudo aborda uma análise comparativa das duas linguagens citadas anteriormente, levando em consideração aspectos de codificação e performance[1], para identificar qual alternativa, entre as duas citadas, é a mais indicada no desenvolvimento de aplicações nativas iOS.

1.2 Objetivos

Esta monografia tem como objetivo a abordagem de conceitos relacionados a Objective-C e Swift, bem como uma análise comparativa dos aspectos de performance, codificação e informações de armazenamento em disco entre estas duas linguagens; tendo como métricas:

- Uso de CPU;
- Memória;
- Armazenamento em disco (não instalado e instalado);
- Tempo de inicialização;
- Quantidade de arquivos gerados;
- Quantidade de código escrito.

1.3 Artefatos

Para fim de validação deste estudo foi desenvolvido um aplicativo de análise de resultados das loterias caixa nas duas linguagens de programação citadas, visando

[1] Conjunto dos resultados obtidos num teste. (PRIBERAM, 2018)

fazer a análise comparativa tendo os aplicativos criados como base. Ambos os aplicativos foram desenvolvidos o mais semelhante possível no que diz respeito a lógica, tipos e estruturas de dados. Tal esforço visou fornecer o resultado mais preciso possível.

A análise da quantidade de arquivos e códigos-fonte não será realizada levando em consideração o aplicativo citado acima. Tal análise será feita através da análise do arquivo gerado pela IDE XCode e pela análise das informações do aplicativo após instalado no dispositivo.

1.4 Estrutura do trabalho

Os capítulos deste trabalho estão estruturados da seguinte forma: neste primeiro capítulo é apresentada uma introdução do estudo informando quais motivações, justificativas e objetivos.

No segundo é apresentada as fundamentações teóricas, com conceitos relacionados ao iOS e ao desenvolvimento de aplicativos, definindo as linguagens Objective-C e Swift, respectivamente.

No terceiro capítulo é detalhada a metodologia utilizada, bem como os detalhes do artefato utilizado para realização da análise comparativa.

No quarto capítulo são apresentadas e detalhadas, caso a caso, as informações recolhidas a partir dos experimentos realizados.

O quinto capítulo finaliza o trabalho apresentando uma conclusão e projeções para possíveis trabalhos futuros.

2 Fundamentação teórica

Os assuntos abordados no presente capítulo detalham os termos utilizados no estudo, tomando como base artigos, livros e documentações oficiais das tecnologias citadas. Os assuntos abordados estão divididos em: Sistema operacional iOS, Objective-C e Swift.

2.1 Sistema operacional iOS

Na apresentação que foi considerada a maior de todos os tempos da Apple, Steve Jobs, fundador da empresa, apresentou o iphone ao mundo em janeiro de 2007. Durante sua trajetória, o iPhone e iPad literalmente redefiniram o mundo da computação móvel. Por trás destes poderosos dispositivos está o sistema operacional iOS. Por meio de constantes atualizações, melhorias e inovações ao longo dos anos, a Apple tornou o iOS uma das plataformas mais ricas em recursos e bem suportadas do mercado.

O iOS possui uma arquitetura de camadas, divididas entre: *Cocoa Touch*, *Media*, *Core services* e *Core OS*, respectivamente, como pode ser visto na Figura 1.

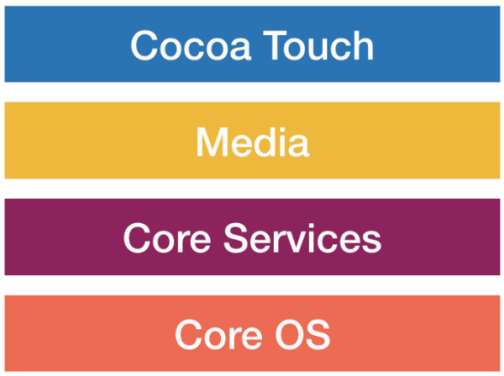

Figura 1 – Arquitetura do sistema operacional iOS
(CALDERA, 2018)

A camanda *Cocoa Touch* contém as *frameworks* relacionadas à interface gráfica e interação com usuário, além de integração com outros serviços da Apple como Mapas, *Game Center*, notificações *push*.

A camada *Media* possui as *frameworks* relacionadas à serviços de áudio e vídeo, além de algumas integrações com o serviço do *iTunes*.

A camada *Core Services* fornece serviços de integração com serviços da Apple como *iCloud* e o *StoreKit*, bem como interface com *hardwares* como localização,

bluetooth, recursos de telefone. Nessa camada estão os mais importantes *frameworks* do iOS que são o *Foundation* e o *Core Foundation*. Eles são os responsáveis pelo gerenciamento de dados e alguns serviços que todos os aplicativos usam, como as coleções, *strings*, *sockets* e as *threads*. (MATTOS B.R.D; BRITTO, 2016)

A camada *Core OS* é a de mais baixo nível do sistema operacional iOS, onde são implementadas todas as integrações a nível de hardware. Todas as funcionalidades de textithardware utilizadas pelas camadas acima, são implementadas pelo *Core OS*.

2.1.1 AppStore e Desenvolvimento de Aplicativos

A primeira versão do iOS, lançado em 2007, não disponibilizava um SDK para desenvolvimento de aplicativo. A preocupação com segurança e controle de qualidade sempre estiveram presentes na história da Apple.

O desenvolvimento de aplicativos iOS é divido em duas fases: implementação e distribuição.

A Apple provê uma IDE proprietária chamada XCode, a qual fornece todas as ferramentas necessárias para desenvolvimento de aplicativos. O primeiro passo da fase de distribuição também é feita através desta ferramenta.

O processo de distribuição de aplicativos envolve uma análise das funcionalidades do aplicativo chamada de revisão. As diretrizes de revisão da App Store são incrivelmente específicas, porém no que se refere à linguagem de programação escolhida para escrever o código fonte, a Apple não oferece restrições. O desenvolvedor pode escolher qualquer linguagem de programação desde que seja gerado um arquivo IPA válido, capaz de ser executado em todos os dispositivos ao qual se destina.

2.2 Objective-C

A linguagem Objective-C foi criada por Brad Cox e sua empresa, a StepStone Corporation, no início da década de 80. Em 1988 ela foi licenciada pela NeXT, tornando-se a linguagem de desenvolvimento do NeXTstep. Atualmente Objective-C é utilizada como a linguagem de programação do MacOS X, que é baseado no NeXTstep. A versão da Apple do ambiente NeXTStep/GNUStep com adições é denominada Cocoa.(DEVMEDIA, 2011)

Objective-C é um superconjunto da linguagem C e fornece recursos orientados a objeto e um tempo de execução dinâmico. Tal linguagem herda a sintaxe, instruções e tipos primitivos de C, adicionando suas particularidades para definir métodos e classes.(APPLE, 2014)

Ao construir aplicativos para o iOS, maior parte do tempo é consumido trabalhando com objetos. Esses objetos são exemplos de classes Objective-C, alguns dos quais são fornecidos pela textitframework Cocoa Touch. Em vez de criar uma classe totalmente nova para fornecer recursos adicionais em uma classe existente, é possível definir uma categoria para adicionar um comportamento personalizado a uma classe existente. Há a possibilidade de usar uma categoria para adicionar métodos a qualquer classe, incluindo classes para as quais não possui o código-fonte de implementação original, como classes de estrutura, por exemplo, *NSString*. Caso tenha o código-fonte original para uma classe, poderá usar uma extensão de classe para adicionar novas propriedades ou modificar as propriedades existentes. As extensões de classe costumam ser usadas para ocultar o comportamento particular para uso em um único arquivo de código-fonte ou na implementação particular de uma estrutura personalizada.

É comum em Objective-C usar as classes do *Cocoa Touch* para representar valores. A classe *NSString* é usada para cadeias de caracteres, a classe *NSNumber* para diferentes tipos de números, como número inteiro ou ponto flutuante, e a classe *NSValue* para outros valores, como estruturas C. Também pode ser usado qualquer um dos tipos primitivos definidos pela linguagem C, como *int*, *float* ou *char*. Em Objective-C coleções são geralmente representadas como instâncias de uma das classes de coleção, como *NSArray*, *NSSet* ou *NSDictionary*, que são usadas para coletar outros objetos.

A maioria do trabalho em um aplicativo Objective-C ocorre como resultado de objetos enviando mensagens uns aos outros. Geralmente, essas mensagens são definidas pelos métodos declarados explicitamente em uma interface de classe. Às vezes, no entanto, é útil poder definir um conjunto de métodos relacionados que não estejam vinculados diretamente a uma classe específica.

Objective-C usa protocolos para definir um grupo de métodos relacionados, como os métodos que um objeto pode chamar em seu textitdelegate, sendo eles opcionais ou obrigatórios. Qualquer classe pode indicar que adota um protocolo, o que significa que também deve fornecer implementações para todos os métodos necessários no protocolo.

2.3 Swift

O Swift é uma linguagem de programação desenvolvida por Chris Lattner e por outros diversos desenvolvedores da Apple, sendo o projeto iniciado em 2010. Trata-se de uma linguagem de propósito geral criada com uma abordagem moderna de padrões de segurança, desempenho e design de software e que se destina a substituir

as linguagens baseadas em C (C, C++ e Objective-C). (APPLE, 2018)

Inicialmente tratava-se de um projeto fechado, porém em Dezembro de 2015 o código Swift tornou-se *open source*. Atualmente está disponível via *GitHub* e inclui suporte para todas as plataformas de software da Apple – iOS, OS X, watchOS e tvOS – bem como para o *Linux*. Os componentes disponíveis incluem o compilador, depurador (*debugger*), bibliotecas padrão, bibliotecas de fundação, gerenciador de pacote e REPL do Swift. O Swift foi licenciado sob a popular licença de código textitlivre Apache 2.0, com uma exceção da biblioteca de tempo de execução (*runtime*) (APPLE, 2018), possibilitando aos usuários incorporar facilmente o Swift em seu próprio software e trasladar a linguagem para novas plataformas.

Projetada visando substituir em médio/longo prazo a linguagem Objective-C, o Swift surgiu como uma proposta para redução da quantidade de código, melhorar aspectos de performance e segurança, além de uma sintaxe mais moderna e amigável a novos desenvolvedores. De fato, durante a apresentação da *WWDC* 2014 a linguagem foi descrita como ”Objective-C, porem sem o volume de código do C”. (METZ, 2014)

A linguagem Swift é idealmente uma linguagem orientada a objetos, possuindo quase todos os conceitos tradicionais desse paradigma como: classes, objetos, herança, polimorfismo, protocolos. Porém, apresenta algumas peculiaridades, como a ausência de classes e métodos abstratos, o que não significa que isso a deixe menos poderosa. O conceito pode ser implementado utilizando protocolos, por exemplo. Ao mesmo tempo que deixa de apresentar alguns conceitos, o Swift apresenta outros bastante poderosos como *extensions* que permitem adicionar comportamentos à classes já existentes, além de *closures* e *subscripts*: que aumentam flexibilidade de desenvolvimento permitindo mesclar com outros paradigmas de programação, como por exemplo, o paradigma funcional.

Swift apresenta todos os tipos de dados tradicionais: numérico, caracteres e booleano, bem como a implementação de estruturas de dados como coleções e dicionários. Entre os tipos de dados mais utilizados estão *Character* e *String* para representação de caracteres e cadeias de caracteres, *Int* para tipos numéricos inteiros, *Float* e *Double* para tipos numéricos de ponto flutuante e *Bool* para valores booleanos. Para representação de estrutura de dados os mais utilizados são *Array* para listas, e *Dictionary* para dicionários ou *hashtables*. Ambas as estruturas podem especificar o tipo do dado contido nelas.

2.4 Trabalhos relacionados

Este material apresenta um estudo sobre um assunto recorrente no meio acadêmico e mercadológico. Ao longo do desenvolvimento deste artigo foram encontrados

materiais que abordam temas semelhantes.

Flaviano Dias (2018) fez uma análise comparativa entre o desenvolvimento de aplicações móveis utilizando Swift e React Native. Em seu estudo ficou evidenciado que o desenvolvimento utilizando Swift é mais performático do que utilizando o Java script do React Native.

(GARCÍA et al., 2015) fizeram uma análise comparativa entre Objective-C e Swift, buscando identificar aspectos de desenvolvimento como facilidade e velocidade de implementação. O estudo concluiu que, de fato, a linguagem Swift oferece uma sintaxe mais simples e amigável, possibilitando uma maior agilidade de aprendizado e implementação. O Swift também aprimorou algumas estruturas como *switches* e *enums*, tornando-os muito mais poderosos.

3 Metodologias

Este capítulo tem como objetivo detalhar os processos e procedimentos realizados para execução da analise comparativa proposta pelo presente trabalho. A análise comparativa em questão envolve duas tecnologias utilizadas para o desenvolvimento de aplicativos para o sistema operacional iOS. Para execução do estudo foi definido um processo o qual foi dividido nas etapas a seguir:

- Definição das linguagens de programação a serem utilizadas para desenvolvimento do artefato, o qual será avaliado seguindo pontos previamente definidos;
- Definição e implementação do artefato a ser desenvolvido, bem como do seu caso de uso;
- Definição e configuração do ambiente de desenvolvimento e dos dispositivos que serão utilizados para executar os testes;
- Definição dos recursos a serem testados e analisados;
- Definição da aplicação que irá realizar a coleta dos dados dos recursos utilizados pelos dispositivos, durante os testes;
- Execução dos testes das aplicações desenvolvidas;
- Análise dos dados recolhidos pela ferramenta previamente definida, observando as informações e comparando as linguagens de acordo com as métricas estabelecidas.

3.1 Artefato e Caso de Uso

Para fim de validação do estudo foi desenvolvido dois aplicativos, um deles utilizando a linguagem Swift e o outro utilizando a linguagem Objective-C. Ambos os aplicativos possuem o mesmo caso de uso: análise de resultados de sorteios das loterias Caixa Econômica Federal. Foram levadas em consideração as loterias:

- Mega-sena;
- Quina;
- Lotomania;
- Lotofacil.

O aplicativo possui embarcado os resultados das loterias acima, desde o primeiro concurso, e oferece as opções: listar resultados, mais sorteados e probabilidades. As três funcionalidades citadas foram implementadas da forma mais semelhante possível nas duas linguagens no que se refere a lógica de implementação. Tipos de dados e estrutura de dados foram utilizadas de acordo com a linguagem. Por mais que o Swift possua os tipos *NSString*, *NSArray* e *NSDictionary*, estes são tipos e estruturas pertencentes ao Objective-C e não ao Swift. Nestes casos, no código Swift, foram utilizadas os tipos *String*, *Array* e *Dictionary*, próprios do Swift.

Os resultados estão em quatro arquivos JSON diferentes porém possuem estrutura semelhantes, conforme Figura 2. A única diferença é na quantidade de bolas sorteadas, que muda de acordo com o sorteio selecionado pelo usuário.

```
{
  "Concurso": 2145,
  "Data": "24/04/2019",
  "bola 1": 6,
  "bola 2": 59,
  "bola 3": 28,
  "bola 4": 8,
  "bola 5": 51,
  "bola 6": 53
},
```

Figura 2 – Estrutura do arquivo JSON

Ao abrir o aplicativo é exibida a tela inicial, com as opções de loterias na parte superior, implementada utilizando o componente *UISegementedControl*, conforme Figura 3. O usuário pode escolher a loteria a qual ele quer obter informações.

As informações disponíveis são listadas logo abaixo em um componente *UITableView*, que possui três opções:

1. **Resultados**: Essa funcionalidade consiste em listar todos os resultados desde o início, separado por concurso.
2. **Mais sorteados**: Essa funcionalidade lista os números mais sorteados, obedecendo à característica da loteria previamente selecionada.
3. **Probabilidades**: Essa funcionalidade exibe a probabilidade, dado o histórico, de cada número daquela loteria ser devidamente sorteado. Agrupando os que possuem probabilidade em comum.

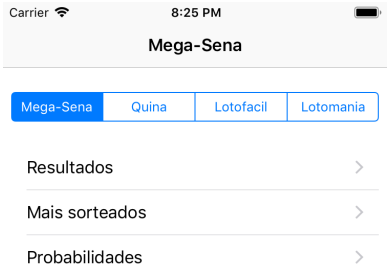

Figura 3 – Tela inicial do aplicativo

- **Resultados**: Ao selecionar essa funcionalidade, o aplicativo utiliza o arquivo de resultados para a loteria selecionada, percorrendo todos os sorteios e listando-os no formato concurso, data e números sorteados. O componente utilizado para exibição das informações é o *UICollectionView*. Conforme exibido na Figura 4.

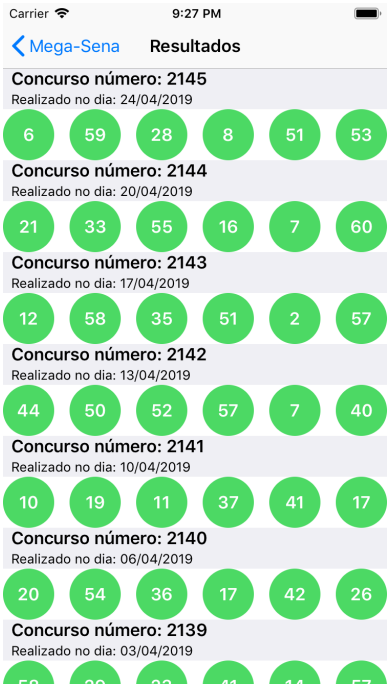

Figura 4 – Tela de resultados

- **Mais sorteados**: Ao selecionar essa funcionalidade, o aplicativo percorre todos

os resultados, ordenando números sorteados mais recorrentemente, de acordo com a quantidade de números sorteados por cada loteria. Tal ordenação foi feita utilizando métodos de ordenação disponibilizados pelas próprias linguagens, que utilizam os algoritmos *MergeSort* no Objective-C e *IntroSort* no Swift. O componente *UICollectionView* é utilizado para listar as informações na tela do dispositivo, como pode ser visto na Figura 5 abaixo.

Figura 5 – Tela de mais sorteados

- **Probabilidades**: Essa funcionalidade percorre os resultados de todos os concursos, executando o cálculo de probabilidade para cada elemento do conjunto de números sorteados. A probabilidade é extraída através do resultado da fórmula descrita na Figura 6.

$$P(A) = \frac{n(A)}{n(U)}$$

Figura 6 – Fórmula para cálculo da probabilidade

.

P(A) representa a probabilidade do evento, *n(A)* o número de casos favoráveis, e *n(U)* o número de casos possíveis. No contexto da funcionalidade *n(A)* representa a quantidade de vezes que um determinado número foi sorteado, *n(U)* representa

o quantidade de números sorteados no espaço amostral. O nosso espaço amostral representa todos os concursos da loteria, desde o início até o concurso 2.145 da mega-sena, 4.958 da quina, 1.805 da Lotofacil e 1.963 da Lotomania.

A tela de probabilidades exibe as informações através de uma *UICollectionView*, separando as probabilidades por seções e agrupando os números de igual probabilidade, conforme Figura 7.

Figura 7 – Tela de probabilidades

.

3.2 Ambiente de desenvolvimento

O artefato e os casos de uso foram pensados e desenvolvidos pelo autor do presente estudo, que possui conhecimentos avançados em desenvolvimento iOS utilizando Swift e intermediários utilizando Objective-C. Os detalhes técnicos dos dispositivos utilizados para desenvolvimento e testes estão listados a seguir:

- **Desenvolvimento:**
  1. **Modelo:** MacBook Pro 2017, 2,3 GHz Intel Core i5, 8 GB, Intel Iris Plus Graphics 640 1536 MB.
  2. **Sistema Operacional:** macOS Mojave v10.14.5
  3. **IDE:** XCode v10.2.1
  4. **Linguagens de programação:** Swift 4.2 e Objective-C

- **Testes:**

    1. **Modelo:** iPhone8 64GB, iPhone XS 128GB
    2. **Sistema Operacional:** 12.3.1

3.3 Metodologia de avaliação

Concluído o desenvolvimento do artefato podemos definir os parâmetros que serão utilizados para fazer a análise, que tem como objetivo a comparação entre as duas linguagens de programação mais utilizadas para o desenvolvimento de aplicativos iOS.

Os dados levantados são resultados da utilização dos recursos computacionais dos dispositivos durante a execução dos testes. Os parâmetros definidos foram a utilização de recursos como: CPU, memória e tempo de inicialização do aplicativo; tamanho do arquivo *xcarchive* e tamanho do aplicativo instalado; e quantidade de arquivos e código desenvolvido.

Foram realizados testes em dois dispositivos, conectados ao macbook através da porta *thunderbolt 3*. A rotina dos testes consistiu em executar 20 vezes cada funcionalidade do aplicativo. A rodada de teste consistiu em:

1. Abrir o aplicativo;
2. Escolha da loteria;
3. Escolher a opção Resultados;
4. Rolar tela de resultados e voltar para tela principal;
5. Escolher a opção Mais Sorteados e voltar para tela principal;
6. Escolher a opção Probabilidades;
7. Rolar a tela de probabilidades;
8. Fechar e forçar encerramento do aplicativo;

Ao final de cada rodada os dados eram anotados para posteriormente serem analisados. A loteria selecionada para execução dos testes foi a quina, por possuir o maior número de dados, tendo ao total 4.958 concursos.

3.3.1 Ferramenta para extração dos dados

A definição de qual ferramenta utilizar para efetuar a extração dos dados é uma etapa muito importante do estudo. Para o nosso experimento optamos por utilizar o *Debug Navigator*, do próprio Xcode. Trata-se de uma seção da IDE que possui integração com outra poderosa ferramenta disponibilizada pela Apple chamada XCode *Instruments*.

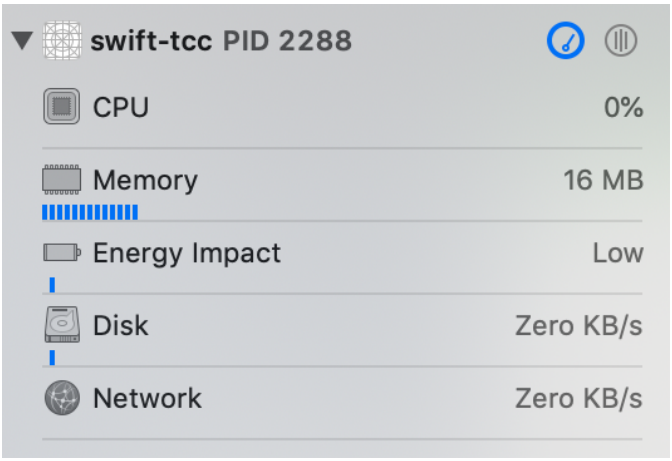

Figura 8 – Debug Navigation

.

O Xcode *Instruments* é uma poderosa e flexível ferramenta de análise de desempenho.(APPLE, 2016) A aplicação fornece diversas ferramentas que permitem fazer medições em diversos aspectos de um aplicativo, desde hardware até recursos de software. Também possui ferramentas que auxiliam no desenvolvimento identificando *memory leaks*, *zombie objects*, contador de referências, entre outros.

Para execução da análise de performance, foi utilizado o *Debug Navigator*, que permite analisar utilização de recursos como CPU (através do *CPU Report*), memória (através do *Memory Report* - Figura 9), bateria (através do *Energy Report*), rede (através do *Network Report*) e disco (através do *Disk Report*).

Para medição do tamanho do aplicativo antes e depois da instalação não utilizamos nenhuma ferramenta específica. O tamanho do aplicativo antes da instalação foi calculado a partir do arquivo *xcarchive* gerado; e para medição do tamanho após a instalação, validamos a informação através do menu de preferências dos dispositivos de teste.

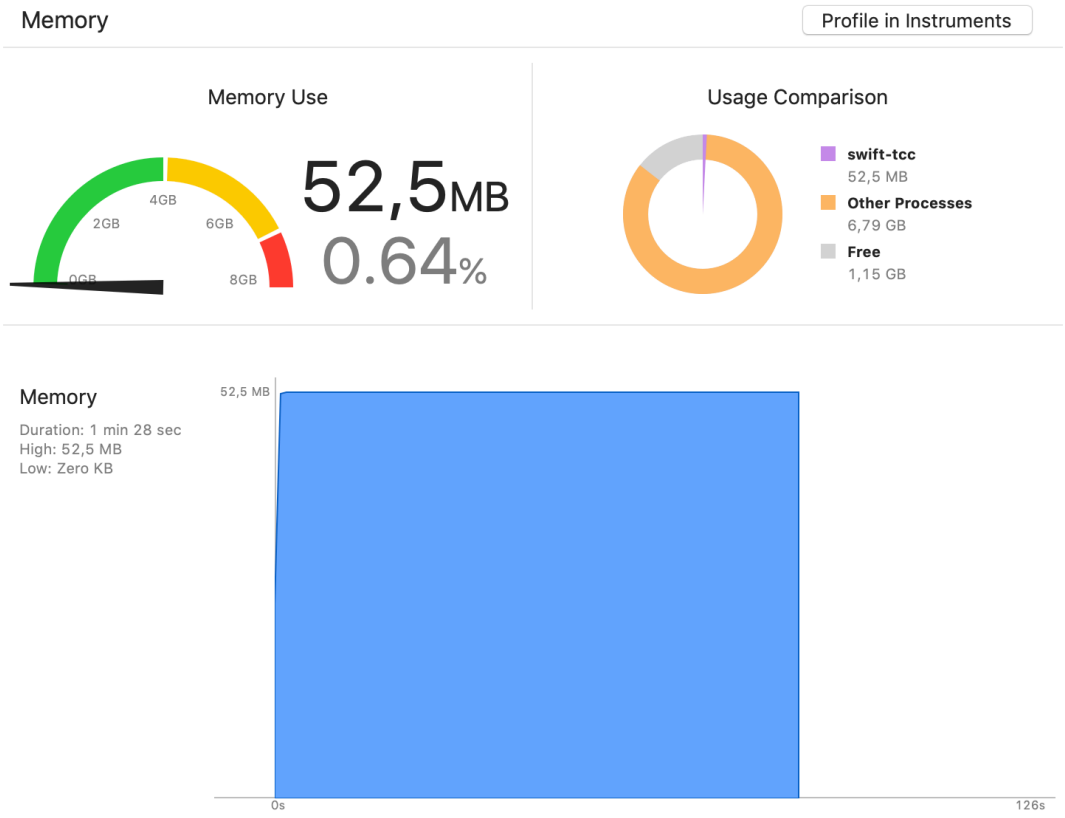

Figura 9 – Memory Report

.

4 Analise comparativa

O presente estudo se propôs a fazer uma análise comparativa entre o desenvolvimento de aplicativos iOS utilizando as duas linguagens de programação mais populares para este fim: Swift e Objective-C. O objetivo principal era responder a pergunta: Qual entre as duas linguagens é a melhor alternativa para desenvolvimento de aplicativos iOS, do ponto de vista de performance e simplicidade.

Baseado no questionamento foram levantadas duas hipóteses:

- **H0:** A linguagem Swift é mais performática do que a linguagem Objective-C, no que tange desenvolvimento de aplicações iOS.
- **H1:** A linguagem Swift requer menos arquivos para desenvolvimento e, consequentemente, menos código implementado.

As hipóteses acima foram definidas considerando as características e particularidades dos aplicativos utilizados na análise em questão.

4.1 Resultados

Esta sessão aborda os resultados obtidos através da coleta dos dados durante a execução dos testes. Iniciamos fazendo a coleta e análise dos dados relacionados à uso de CPU e memória, seguido do armazenado ocupado pelos arquivos pré e pós instalação, tempo de inicialização dos aplicativos e por último quantidade de arquivos e código escrito.

4.1.1 Análise de CPU

A atividade da CPU é um parâmetro muito importante em desenvolvimento de aplicações móveis, pois é um termômetro para alguns fatores chave. Quanto menor o uso de CPU mais otimizado é uma determinada aplicação, o que implica em uma menor temperatura do dispositivo e consequentemente menor consumo de bateria.

A primeira análise foi a de consumo de CPU, por incorporar esses três fatores chaves citados acima: otimização, temperatura e consumo. Para medir o uso deste recurso foi utilizada a ferramenta de relatório *CPU Report* da própria IDE Xcode.

Os dados foram coletados manualmente a medida que as rodadas de testes eram executadas. A comparação tomou como métrica os valores médios obtidos, conforme Tabela 1 e Figura 10.

Funcionalidades	Swift	Objective-C
Selecionar Loteria	9,7%	8,4%
Resultados	12,4%	12,9%
Rolar página de resultados	31,8%	29,3%
Mais sorteados	9,9%	15,0%
Probabilidades	11,4%	15,9%
Rolar página de probabilidades	21,5%	22,5%
Média:	16,1%	17,3%

Tabela 1 – Representação dos dados de uso médio de CPU

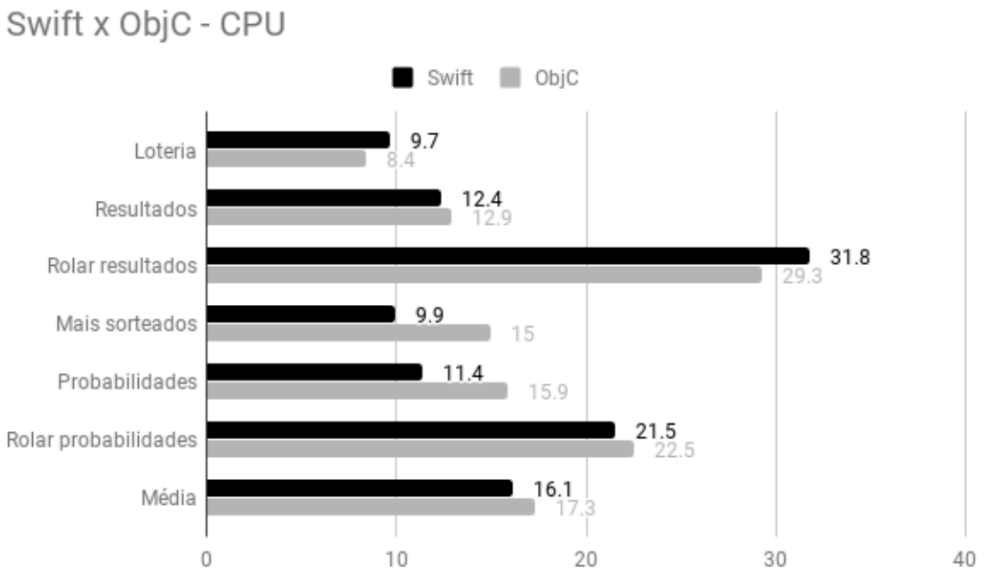

Figura 10 – Representação gráfica do uso médio de CPU por funcionalidade

.

No que tange uso de CPU o desenvolvimento utilizando a linguagem Swift apresentou uma pequena vantagem quanto ao desenvolvimento utilizando Objective-C.

No geral o Swift apresentou melhor desempenho no uso da CPU, do que o Objective-C.

4.1.2 Uso de memória

A utilização de memória é outro parâmetro muito importante em desenvolvimento de aplicações móveis. Por possuir uma quantidade reduzida, quando comparado a dispositivos maiores como *desktops* e *notebooks*, a otimização do uso da memória do dispositivo provê melhor tempo de resposta, o que causa a sensação de melhor fluidez nos aplicativos.

A Tabela 2 apresenta o resultado das rodadas de testes relacionados ao consumo de memória. Os dados apresentados representam o consumo médio de memória utilizada em ***MB*** após as 20 rodadas de teste. A representação gráfica pode ser vista

na Figura 11.

Funcionalidades	Swift	Objective-C
Selecionar Loteria	18,2	15,3
Resultados	24,01	22,56
Rolar página de resultados	28,16	26,61
Mais sorteados	21,24	18,405
Probabilidades	21,755	18,96
Rolar página de probabilidades	23,53	20,775
Média:	22,8	20,4

Tabela 2 – Uso médio de Memória (valores em *megabytes*)

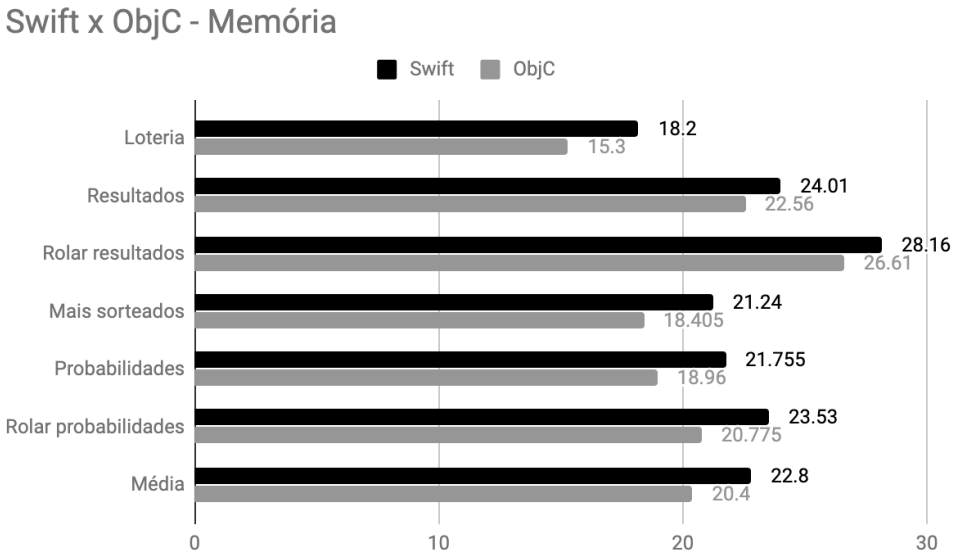

Figura 11 – Representação gráfica do uso médio de memória por funcionalidade

.

O Objective-C apresentou uma melhor performance na utilização de memória em todas as funcionalidades. Isso acontece porque a *UIKit*, *framework* responsável por prover os componentes de interação com usuário, é desenvolvido em Objective-C. Quando o desenvolvedor escolhe desenvolver em Swift, internamente, são criados *bridges* para permitir que métodos Objective-C possam ser invocados pelo código Swift. Isso faz com que, diferentemente do caso do Objective-C, o compilador não faça uma otimização do código no momento da geração do binário.

4.1.3 Armazenamento

O armazenamento foi analisado de duas formas: tamanho pré-instalação e pós instalação. A primeira envolveu analisar o tamanho do arquivo após o processo de *archive* do XCode. O arquivo gerado possui o formato *xcarchive*, que contem o app e

arquivos utilizados pela loja para diversos fins. Conforme pode ser visto na Tabela 3 e pela figura 12, o Swift gera um *.xcarchive* 59,8 vezes menor do que o arquivo gerado em Objective-C. Isso ocorre porque o Swift é empacotado junto com o aplicativo, o que aumenta o tamanho do arquivo gerado.

Linguagem	Arquivo *xcarchive*
Objective-C	5,1mb
Swift	305mb

Tabela 3 – Armazenamento - Arquivo *xcarchive*

A segunda forma envolveu analisar o tamanho dos arquivos pós-instalação nos dispositivos. O aplicativo feito utilizando Objective-C ocupou menos espaço após instalado, conforme pode ser visto na Tabela 4 e na figura 12. O aplicativo feito em Swift ocupou 17,9 vezes mais espaço do que o aplicativo feito em objetive-c.

Linguagem	Aplicativo instalado
Objective-C	2,6mb
Swift	46,6mb

Tabela 4 – Armazenamento - Aplicativo instalado

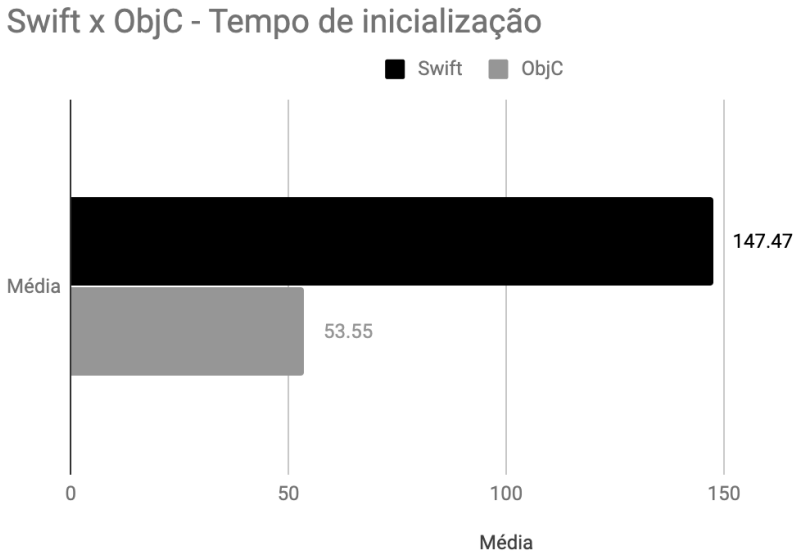

Figura 12 – Representação gráfica do armazenamento do arquivo *xcarchive* e pós instalação

4.1.4 Tempo de inicialização

O tempo de inicialização do aplicativo pode ser calculada através de uma variável de ambiente habilitada no XCode chamada *DYLD_PRINT_STATISTICS*. Com

essa variável habilitada é possível analisar o tempo de resposta de cada um dos componentes responsáveis por inicializar o aplicativo, conforme pode ser visto na Figura 13.

```
Total pre-main time: 256.37 milliseconds (100.0%)
         dylib loading time: 183.44 milliseconds (71.5%)
        rebase/binding time: 411015771.6 seconds (14971307.3%)
            ObjC setup time:  41.78 milliseconds (16.2%)
           initializer time:  42.84 milliseconds (16.7%)
           slowest intializers :
             libSystem.B.dylib :   7.35 milliseconds (2.8%)
   libMainThreadChecker.dylib :  14.34 milliseconds (5.5%)
 libViewDebuggerSupport.dylib :  13.81 milliseconds (5.3%)
```

Figura 13 – *Output* do tempo de inicialização.

A média do tempo de inicialização apresentado pelo aplicativo em Objective-C foi muito melhor do que o do aplicativo em Swift. Conforme pode ser visto na Tabela 5 e na figura .

	Objective-C	Swift
Média em *ms*	53,5465	147,4745

Tabela 5 – Tempo de inicialização em *ms*

O aplicativo feito utilizando Objective-C apresentou um tempo de inicialização 2,75 vezes mais rápido do que o aplicativo em Swift.

4.1.5 Quantidade de arquivos

A quantidade de arquivos foi comparada levando em consideração todos os arquivos do projeto: arquivos de código, configuração e dados. É importante salientar que os dois projetos foram feitos utilizando a mesma estrutura de arquivos e arquitetura.

Para o projeto em objective-c foram necessários 35 arquivos, enquanto o projeto Swift precisou de 24 arquivos, conforme Figura 14. Isso ocorre porque, de fato, o Swift foi desenvolvido visando reduzir o número de arquivos necessários para o desenvolvimento.

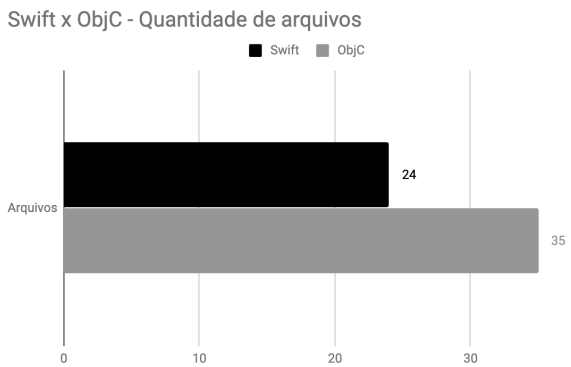

Figura 14 – Representação gráfica da quantidade de arquivos

Objective-C se assemelha mais ao C, em que é necessário um arquivo *header (.h)* para criar interfaces públicas para os *modules (.m)*. O Swift possui o arquivo *.swift* no qual é possível implementar protocolos, classes e textitextensions.

O objective-C também cria automaticamente o arquivo *main.m* responsável por inicializar a aplicação, enquanto o Swift faz isso de forma transparente.

A tabela com os arquivos dos projetos em cada linguagem pode ser encontrada no apêndice ao final deste trabalho.

4.1.6 Quantidade de código

Para comparar a quantidade de código foi definido que apenas os códigos criados pelo desenvolvedor seriam comparados. Logo, não foi avaliado as classes criadas automaticamente como *main*, *AppDelegate*, *Info.Plist*, já que não houveram alterações nestas classes por parte do desenvolvedor.

Os arquivos *.h* e *.m* do Objective-C foram somados e comparados ao arquivo *.swift* da linguagem Swift.

A linguagem objective-C apresentou mais caracteres por arquivo do que o Swift, conforme pode ser visto na Tabela 6.

De fato, a sintaxe do Swift é muito mais amigável não precisando fazer uso de caracteres que são obrigatórios no objective-c, assemelhando-se a linguagens mais modernas como Python e Ruby.

	Objective-C	Swift
ViewController	3644	2932
Repository	1359	803
Theme	1550	1010
ResultsPresenterDelegate	656	597
AllResultsPresenter	1693	1196
MostDrawnNumbersPresenter	2403	1802
ProbabilitiesPresenter	3564	2557
ResultViewController	3297	3186
BallCollectionViewCell	1085	687
HeaderCollectionReusableView	1108	651
Total:	20359	15421

Tabela 6 – Quantidade de caracteres

5 Conclusão e Trabalhos futuros

O presente trabalho teve como objetivo avaliar os resultados da análise entre as linguagens Objective-C e Swift. Os resultados deste estudo foram mais promissores para a linguagem Objective-C, porém com alguns aspectos favoráveis ao Swift.

A linguagem Swift apresentou vantagens quanto a utilização de CPU, quantidade de arquivos e código. Tais vantagens apresentadas não foram tão expressivas quanto algumas vantagens apresentadas pelo Objective-C.

O Objective-C apresentou vantagens quanto a utilização de memória, tempo de inicialização e armazenamento, sendo estas duas últimas vantagens expressivas. O tempo de inicialização do aplicativo escrito em Objective-C foi 2,75 vezes mais rápido do que o aplicativo escrito em Swift. O arquivo *xcarchive*, gerado utilizando Objective-C foi 59,8 vezes menor do que o arquivo gerado utilizando Swift. Após a instalação o aplicativo feito utilizando Objective-C ocupou 17,9 vezes menos do que o aplicativo feito em Swift.

De maneira geral podemos considerar que o desenvolvimento utilizando Swift não apresentou as vantagens que eram esperadas. Pode-se concluir que há uma pequena vantagem performática no desenvolvimento utilizando Objective-C, que apresentou melhor utilização de memória, melhor tempo de inicialização e menor necessidade de armazenamento quanto aos arquivos gerados. Enquanto o Swift apresentou uma melhor utilização de CPU.

Quanto aos pontos que não envolvem performance e que foram avaliados o Swift foi superior, gerando menos arquivos de código-fonte e menos caracteres para desenvolvimento do código.

É muito importante salientar que o resultado acima se aplica para este estudo, e que deve ser considerado as peculiaridades dos artefatos desenvolvidos e a metodologia de teste aplicada. Especificamente neste caso a linguagem Objective-c se mostrou superior quando comparada a linguagem Swift.

5.1 Dificuldades encontradas

As dificuldades encontradas para desenvolvimento do presente trabalho envolveram principalmente:

- A disparidade de conhecimento do desenvolvedor quanto as duas linguagens. O autor possui mais experiência e conhecimento em desenvolvimento Swift do que

Objective-C.

- Não possibilidade de aferir o consumo energético. A ferramenta disponibilizada pelo XCode não apresenta detalhes suficientes para tirar conclusões claras sobre a utilização desse recurso.
- A não automação dos testes. Por possuir caráter manual, por mais que o autor tenha tentado executar da forma mais semelhante possível, os resultados podem ter sofrido algum tipo de impacto.

5.1.1 Trabalhos futuros

Os trabalhos futuros podem seguir o mesmo objetivo deste. Cada atualização de versão do Swift disponibilizada, inclui algumas melhorias de performance. A análise comparativa pode ser atualizada conforme as linguagens forem recebendo atualizações.

Durante o desenvolvimento deste estudo a Apple anunciou o *Swift UI*, que possui a *framework* de *UI* atualizada. Diferentemente da *UIKit* o *Swift UI* é escrita em Swift. Este é um fator importante e que merece ser analisado comparativamente, pois pode representar uma melhoria do Swift quanto a utilização de memória, por exemplo, já que *UI* é parte majoritária em aplicativos de uso geral.

Outro ponto a ser estudado é analise das estruturas de dados isoladamente, comparando a performance de cada operação.

6 Apêndice

6.1 Tabelas referentes aos testes e comparações

	Objective- C	**Swift**
Teste 1	131,51	149,61
Teste 2	103,11	146,04
Teste 3	102,16	147,86
Teste 4	142,06	152,71
Teste 5	63,56	146,20
Teste 6	28,48	147,86
Teste 7	33,45	148,11
Teste 8	32,86	145,14
Teste 9	35,23	141,99
Teste 10	59,44	145,16
Teste 11	30,13	151,36
Teste 12	41,05	142,75
Teste 13	33,98	144,81
Teste 14	35,97	143,85
Teste 15	33,09	143,98
Teste 16	34,13	145,07
Teste 17	36,92	147,85
Teste 18	28,32	153,64
Teste 19	28,85	151,15
Teste 20	36,66	154,35
MÉDIA	**53,5465**	**147,4745**
DESVIO PADRÃO	**36,02**	**3,64**
VARIÂNCIA	**1233,15**	**12,59**

Tabela 7 – Teste de tempo de inicialização em milissegundos

	Loteria	**Resultados**	**Rolar resultados**	**Mais sorteados**	**Probabilidades**	**Rolar Probabilidades**
1	10	13	30	11	12	23
2	10	13	39	10	10	20
3	10	13	32	9	12	19
4	10	14	29	11	12	21
5	10	14	29	10	13	26
6	9	14	37	9	12	25
7	10	10	27	11	11	23
8	10	12	30	10	12	17
9	10	13	31	8	11	23
10	8	12	34	11	12	22
11	9	14	33	12	12	21
12	10	11	30	9	13	20
13	9	11	34	9	12	18
14	10	12	34	9	9	21
15	10	13	34	9	10	18
16	10	15	26	10	12	26
17	9	12	31	10	12	22
18	10	12	30	9	7	22
19	10	8	35	9	13	20
20	10	12	31	11	11	22
MÉDIA:	**9,7**	**12,4**	**31,8**	**9,9**	**11,4**	**21,5**
DESVIO PADRÃO:	**0,57**	**1,6**	**3,22**	**1,03**	**1,46**	**2,5**
VARIÂNCIA:	**0,31**	**2,44**	**9,86**	**1,02**	**2,04**	**5,94**

Tabela 8 – Médias percentuais do teste utilização de CPU - Swift

	Loteria	**Resultados**	**Rolar resultados**	**Mais sorteados**	**Probabilidades**	**Rolar Probabilidades**
1	8	12	30	15	17	18
2	9	14	25	14	16	24
3	9	15	32	16	16	23
4	9	13	32	15	17	23
5	9	14	29	14	17	26
6	9	14	29	14	17	21
7	7	10	28	16	18	19
8	9	13	33	15	15	27
9	8	12	29	15	15	23
10	8	12	29	14	11	26
11	8	7	29	16	17	25
12	9	14	26	15	17	19
13	9	13	28	15	18	25
14	8	14	31	14	15	20
15	8	14	30	15	15	24
16	8	11	28	14	12	22
17	8	13	30	16	17	22
18	9	15	29	16	15	23
19	8	14	31	16	17	19
20	8	14	28	15	16	21
MÉDIA:	**8,4**	**12,9**	**29,3**	**15**	**15,9**	**22,5**
DESVIO PADRÃO:	**0,59**	**1,89**	**1,95**	**0,79**	**1,80**	**2,62**
VARIÂNCIA:	**0,0034**	**0,033**	**0,0036**	**0,006**	**0,03**	**0,065**

Tabela 9 – Médias percentuais do teste utilização de CPU - Objective-C

	Loteria	**Resultados**	**Rolar resultados**	**Mais sorteados**	**Probabilidades**	**Rolar Probabilidades**
1	17,9	23,8	28,2	21	21,6	22,8
2	17,9	25,4	28,8	21,3	21,7	23,4
3	18,6	24,5	28,4	21,3	21,9	23,5
4	18,1	24,6	27,9	21,1	21,7	23,3
5	17,9	23,9	27,8	21,2	21,7	23,4
6	19	24,4	28,6	22,1	22,6	24,3
7	17,8	23,3	26,4	21	21,6	23,6
8	18,9	24,7	29	21,5	22,1	23,5
9	18,9	24,2	28,6	21,7	22,2	23,9
10	17,9	23,9	28	21,1	21,5	22,8
11	18,1	23,8	28,2	21,4	22	24
12	18	23,5	28	21,8	21,7	24,1
13	18,2	23,7	28,3	20,9	21,4	23,3
14	18	23,6	28,1	21,3	22	24,2
15	17,3	22,9	27,6	20,7	21,2	23,3
16	18,1	23,8	27,8	21,2	21,8	23,6
17	20,8	24,5	28,9	21,9	22,4	24
18	18,3	27,4	29,2	21,1	21,7	23,5
19	18,1	23,7	28,1	21	21,6	23,4
20	17,1	23,3	27,3	20,2	20,7	22,7
MÉDIA:	**18,2**	**24,1**	**28,16**	**21,24**	**21,755**	**23,53**
DESVIO PADRÃO:	**0,577**	**1,0**	**0,63**	**0,43**	**0,41**	**0,45**
VARIÂNCIA:	**0,22**	**0,9**	**0,38**	**0,17**	**0,16**	**0,19**

Tabela 10 – Teste utilização de Memória - Swift (valores em *megabytes*)

	Loteria	**Resultados**	**Rolar resultados**	**Mais sorteados**	**Probabilidades**	**Rolar Probabilidades**
1	15,5	22,6	26,9	18,4	18,9	20,9
2	15,3	22,5	27,2	18,4	19	20,6
3	15,2	22,5	26,3	18,3	18,9	20,6
4	15,2	22,6	26,6	18,3	19	21,1
5	16,2	22,5	26,5	18,2	18,6	20,7
6	15,3	22,7	27,3	18,6	19,1	20,5
7	15,3	22,6	27,2	18,6	19,2	21,2
8	15,1	22,4	26,8	18,3	19	20,4
9	15,1	22,4	25,7	18,2	18,8	20,7
10	15,1	22,5	25,8	18,3	19	20,5
11	15,3	22,6	26,4	18,6	19	20,9
12	15,2	22,5	26,4	18,6	19	20,4
13	15,4	22,6	26,2	18,5	19,1	20,8
14	15,1	22,5	26,9	18,4	18,9	21,1
15	15,2	22,6	26,5	18,4	19	20,8
16	15,3	22,6	26,9	18,6	19,1	20,6
17	15,2	22,6	26,9	18,4	19	20,9
18	15,1	22,6	26,9	18,2	18,8	21,2
19	15,2	22,7	25,8	18,4	18,9	20,6
20	15,2	22,6	27	18,4	19	21
MÉDIA:	**15,275**	**22,56**	**26,61**	**18,405**	**18,965**	**20,775**
DESVIO PADRÃO:	**0,242**	**0,082**	**0,475**	**0,139**	**0,130**	**0,255**
VARIÂNCIA:	**0,055**	**0,006**	**0,214**	**0,018**	**0,016**	**0,061**

Tabela 11 – Teste utilização de Memória - Objective-C (valores em *megabytes*)

Objective-C	**Swift**
resultados-lotofacil.json	resultados-lotofacil.json
resultados-lotomania.json	resultados-lotomania.json
resultados-mega-sena.json	resultados-mega-sena.json
resultados-quina.json	resultados-quina.json
Assets.xcassets	Assets.xcassets
LaunchScreen.storyboard	LaunchScreen.storyboard
Info.plist	Info.plist
main.m	AppDelegate.swift
AppDelegate.h	ViewController.swift
AppDelegate.m	Main.storyboard
ViewController.h	Repository.swift
ViewController.m	Theme.swift
Main.storyboard	ResultPresenterDelegate.swift
Theme.h	AllResultsPresenter.swift
Theme.m	MostDrawnNumbersPresenter.swift
Repository.h	ProbabilitiesPresenter.swift
Repository.m	ResultViewController.swift
ResultPresenterDelegate.h	ResultViewController.xib
AllResultsPresenter.h	BallCollectionViewCell.swift
AllResultsPresenter.m	BallCollectionViewCell.xib
MostDrawnNumbersPresenter.h	HeaderCollectionReusableView.swift
MostDrawnNumbersPresenter.m	HeaderCollectionReusableView.xib
ProbabilitiesPresenter.h	swift-tcc.app
ProbabilitiesPresenter.m	
ResultsViewController.h	
ResultsViewController.m	
ResultsViewController.xib	
BallCollectionViewCell.h	
BallCollectionViewCell.m	
BallCollectionViewCell.xib	
HeaderCollectionReusableView.h	
HeaderCollectionReusableView.m	
HeaderCollectionReusableView.xib	
objc-tcc.app	

Tabela 12 – Arquivos do aplicativo Objective-C e Swift

Referências

APPLE. *About Objective-C*. 2014. Disponível em: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>. Citado na página 16.

APPLE. *Instruments Help Topics*. 2016. Disponível em: <https://developer.apple.com/library/archive/documentation/AnalysisTools/Conceptual/instruments_help-collection/Chapter/Chapter.html>. Citado na página 26.

APPLE. *About swift*. 2018. Disponível em: <https://swift.org/about/>. Citado na página 18.

CALDERA, A. *Camadas do sistema operacional iOS*. 2018. Disponível em: <https://medium.com/@anuradhs/ios-architecture-a2169dad8067>. Citado na página 15.

DEVMEDIA. *Introdução ao Objective-C*. 2011. Disponível em: <https://www.devmedia.com.br/introducao-ao-objective-c/23061>. Citado na página 16.

DIAS, F. *Análise comparativa no desenvolvimento de aplicações móveis utilizando Swift e React Native*. 2018. Disponível em: <https://drive.google.com/file/d/12rsMqsOKdDTbfAYgFVxkKN8yAOC47BqF/view>. Citado na página 19.

GARCÍA, C. et al. Swift vs. objective-c: A new programming language. *International Journal of Interactive Multimedia and Artificial Intelligence*, v. 3, n. 3, p. 74–81, 2015. Disponível em: <https://www.ijimai.org/JOURNAL/sites/default/files/files/2015/05/ijimai20153_3_10_pdf_19818.pdf>. Citado na página 19.

GROTHAUS, M. *The number of apps on Apple's App Store has shrunk for the first time*. 2018. Disponível em: <https://www.fastcompany.com/40554728/the-number-of-apps-on-apples-app-store-has-shrunk-for-the-first-time>. Citado na página 12.

HOLST, A. *Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018*. 2019. último acesso em 22 de maio de 2019. Disponível em: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>. Citado na página 12.

MATTOS B.R.D; BRITTO, J. G. d. Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataformas. 2016. Disponível em: <https://fga.unb.br/articles/0001/5113/Beatriz_Joao_TCC_Aplicativos_M_veis.pdf>. Citado na página 16.

METZ, R. *Apple Seeks a Swift Way to Lure More Developers*. 2014. último acesso em 23 de Junho de 2018. Disponível em: <https://www.technologyreview.com/s/527821/apple-seeks-a-swift-way-to-lure-more-developers/>. Citado na página 18.

PRIBERAM. *Significado/definição de Performance*. 2018. Último acesso em 21 de julho de 2019. Disponível em: <https://dicionario.priberam.org/performance>. Citado na página 13.