UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Visión Electrónica
Más que un estado sólido
http: revistas.udistrital.edu.co/ojs/index.php/visele/index

VISIÓN ELECTRÓNICA

VISIÓN DE CASO

Implementaciones móviles sobre JVM: Lenguajes dinámicos versus lenguajes estáticos

Mobile implementations on jvm: static vs dynamic languages

Miguel Ángel Leguizamón P.[1]***, Juan Camilo Sosa S.***[2]***, Leidy Marcela Herrera C.***[3]

INFORMACIÓN DEL ARTÍCULO

Historia del artículo:
Enviado: 10/10/2015
Recibido: 20/10/2015
Aceptado: 03/11/2015

Palabras clave:
Android
Groovy
Lenguajes dinámicos
Móviles
Nuevas tecnologías

Keywords:
Android
Groovy
Dynamic languages
Mobile
New technologies

RESUMEN

La implementación de aplicaciones móviles está evolucionando aceleradamente, por lo que se requiere optimizar el proceso de desarrollo de estos tipos de software. No obstante, Android es una de las plataformas más aceptadas para aplicaciones en tecnologías móviles, aunque presenta demoras en la ejecución de tareas pues el emulador virtual consume excesivos recursos físicos. Por lo anterior, lenguajes como Groovy ofrecen la versatilidad de programar para JVM (Java Virtual Machine), haciéndose un lenguaje dinámico que permite la optimización de tiempos de desarrollo y la reducción de líneas de código. El presente artículo presenta los resultados de la investigación que condujo a evidenciar los tiempos de desarrollo de Groovy frente a los de Java al probar una aplicación codificada que corre en memoria, con ingreso de texto, almacenamiento temporal, filtro, búsqueda y consulta de listas. El resultado muestra una reducción temporal de construcción y desarrollo que impacta el corrimiento, el cual es cercano al 33 %, asunto que puede resultar favorable en tareas de mayor envergadura.

ABSTRACT

The implementation of mobile applications is evolving rapidly, so it is required to optimize the development process of this type of software. Nevertheless, Android is one of the most widely accepted technologies for applications on mobile platforms, it presents delays in the execution of the tasks, as the virtual emulator consumes excessive physical resources. Therefore, languages like Groovy offer the versatility to program for JVM (Java Virtual Machine) in a dynamic language, allowing optimization of development time and reducing lines of code. In this article, the investigation carries out to evidence, development time versus Groovy Java, displayed when testing a coded application that runs in memory, with text entry, temporary storage, filter, search and query lists; so, the result shows a temporary reduction of construction and development impacting shifting close to 33 %; matter which may be favorable in larger tasks.

[1]Ingeniero de Sistemas, especialista en gerencia de Sistemas Informáticos, MSc. en ciencias de la información y las comunicaciones, docente de planta Universidad Distrital Francisco José de Caldas, Colombia. Correo electrónico: maleguizamonp@correo.udistrital.edu.co.

[2]Tecnólogo en sistematización de datos, Universidad Distrital Francisco José de Caldas; consultor y desarrollador Java Oracle: RTI SAS. Correo electrónico: juank-milo1805@hotmail.com

[3]Tecnóloga en sistematización de datos, Universidad Distrital Francisco José de Caldas; analista de requerimientos: Telefónica. Correo electrónico: faidy713@gmail.com

Citar este artículo como: M. A. Ángel L, J. C. Sosa S, L. M. Herrera C. "Implementaciones móviles sobre JVM: Lenguajes dinámicos versus lenguajes estáticos ". Visión Electrónica, algo más que un estado sólido, Vol. 10, No. 1, 76-82, enero-junio 2016.

1. Introducción

Las tecnologías móviles son aplicadas a tareas múltiples que se logran ejecutar de manera ágil para vencer limitaciones de tiempo y espacio; lo anterior debido a la facilidad de movimiento y ubicación que otorgan las comunicaciones con aplicaciones livianas. Para este fin se pueden usar los lenguajes dinámicos ya que permiten economizar código, integrar funciones y lograr un enfoque más claro partiendo de bases sencillas obteniéndose buenos resultados, [1]; al usar metodologías flexibles que pueden integrarse a desarrollos en Java, las aplicaciones resultan rápidas, ágiles y óptimas, tal es el caso de la metodología Scrum, que otorga la posibilidad de realizar desarrollo en menor tiempo y de alta calidad [2].

Por otro lado, las aplicaciones actuales desarrolladas para tecnologías móviles están implementadas en Android; sin embargo, se debe tener en cuenta que presentan retardos en el tiempo de desarrollo, por lo que mejorar los tiempos de estos procesos es un problema vigente. En tanto, Groovy es un lenguaje de programación no fuertemente tipado (no exige que la variable lleve un tipo de dato), orientado por objetos, soportado en Java y corre sobre la JVM, por lo cual, puede usarse directamente en cualquier aplicación Java; debido a su bajo nivel de acoplamiento y reducción de líneas de código en comparación con otros lenguajes, es fácil de usar y es lo suficientemente robusto para desarrollar aplicaciones.

En consecuencia, el presente artículo pretende mostrar cómo Groovy disminuye el tiempo de desarrollo de aplicaciones Android; para ello, es necesario exponer una comparación de las tecnologías de desarrollo móviles en el último lustro y, a través de una aplicación básica (inserción y búsqueda en listados en memoria), medir el tiempo de desarrollo entre Groovy y Java, entendido como: construcción del paquete apk a partir de la compilación del código fuente y ejecución de la aplicación. La compilación se hace con Android Studio y el tiempo de construcción en cada caso, se mide en segundos.

La estructura del artículo es la siguiente: conceptos básicos sobre lenguajes dinámicos; estado de arte (blogs) que muestran estudios similares y sus resultados utilizando Groovy, pero comparados con Java usando loops; listas y definición de variables: materiales y métodos, problema central, codificación, compilación, medición de códigos; resultados de la comparación y discusión; conclusiones y perspectivas.

2. Conceptos básicos

Para obtener desarrollos ágiles en pocas líneas de código, se utilizan lenguajes dinámicos (LD) que permiten la integración de diversos componentes con otros lenguajes de diferentes tipos, y que pueden manipular el envío de mensajes de manera inmediata, sin importar lo que se está intentando interpretar. A diferencia del tipo de dato estático, los tipos de datos dinámicos evitan extenderse en el desarrollo de código, son prácticos y brindan facilidades en tiempos de desarrollo debido a la indexación de métodos funcionales, optimización de código y reducción de componentes. Se entiende entonces que es más efectivo el manejo de lenguajes dinámicos, ya que estos suministran facilidades sin dejar a un lado la seguridad que implica el desarrollo de software.

Para poder entender el enfoque de los lenguajes dinámicos, a continuación se relacionan algunos conceptos básicos.

Las tecnologías móviles[1] permiten simplificar actividades de la vida diaria y tomaron fuerza con la creación del teléfono móvil, el cual facilitó las comunicaciones a largas distancias de manera práctica por la portabilidad del servicio de telefonía, así como la infraestructura de soporte y servicios, fundamental en la sociedad actual, [3]. Contribuyó a esto Android[2], sistema operativo semejante a IOS, Symbian o Blackberry, [4], cuya diferencia estriba en que está basado en una plataforma totalmente libre como lo es Linux, su estructura está compuesta por aplicaciones que se ejecutan en una framework Java de aplicaciones orientadas a objetos que pueden ejecutarse en varias plataformas, sin necesidad de realizar cambios, lo cual lo hace versátil. Es inusitado el uso e impacto de las tecnologías móviles en las tecnologías de la información actuales, [5].

De otro lado, es pertinente considerar la ventaja de los lenguajes dinámicos sobre los estáticos; básicamente, los lenguajes dinámicos determinan la validez de las operaciones durante tiempo de ejecución, estos permiten que una misma variable asocie diversos valores en diferentes tiempos de ejecución, escribir bloques de código más cortos, alterar el curso del lenguaje de manera legítima, ya que puede inyectarse el código o ejecutar

[1] Son un medio de comunicación que ha superado a la telefonía fija, esto se debe a que las redes de telefonía móvil son más fáciles y baratas de desplegar

[2] Android es un sistema operativo inicialmente pensado para teléfonos móviles [4]

funcionalidades especiales y cambiar variables, aunque suelen tener menos claridad y son propensos a errores [6]. En tanto, los lenguajes estáticos manejan operaciones explícitas donde se determinan las posiciones textuales o puede darse que el compilador infiera en las expresiones y declaraciones según el contexto en el que se encuentre; por lo tanto, es difícil cambiar una variable, el código es legible y permite entender los programas fácilmente, pero pueden detectar y prevenir errores en tiempo de compilación, aunque su código suele ser largo. Java se escribe de forma estática mientras que Groovy, Python, Ruby y JavaScript se escriben dinámicamente; Groovy es un lenguaje dinámico, pero a diferencia de Php o Python (también dinámicos) corre sobre la JVM (Java Virtual Machine) por lo que facilita encontrar errores de codificación en el momento de ejecutar la aplicación [7].

Por otra parte, Groovy es un lenguaje orientado por objetos que valida código de Java; además, es un lenguaje dinámico que no requiere que se especifiquen los tipos de referencias, la ventaja de esto es que permite modificar desde métodos hasta objetos en tiempo de ejecución; su sintaxis es más blanda que Java y contiene soporte para listas, mapas y closures en forma nativa, [8]. A diferencia de Java, que requiere hacer importaciones de librerías, Groovy permite hacer uso de clases y métodos de manera nativa, pues estos están embebidos dentro del lenguaje mismo, esto lo hace un poco más pesado, [9].

Estos son los paquetes incluidos en Groovy que no están en Java de manera nativa, [10].

- java.io.*
- java.net.*
- java.util.*
- groovy.lang.*
- groovy.util.*

En consecuencia, realizar aplicaciones en Groovy simplifica la manera de codificar, ahorrando tiempo de desarrollo, [11], como se muestra en la figura **1**.

3. Estado del arte

Según la página TIOBE, en el listado de los 50 lenguajes de programación más usados se puede observar que Java aparece en el primer lugar con 19.543 %, Groovy aparece en un lugar más lejano, en la posición 32, con un 0.583 %, [12]; sin embargo, este ranking no permite ver el potencial de Groovy, puesto que este ha venido creciendo poco a poco y, actualmente, existen variadas herramientas orientadas al desarrollo en este lenguaje (Grails, Griffon) y se adiciona la característica de soporte para Android. Por esto, ahora es uno de los primeros lenguajes que corren sobre la JVM que los desarrolladores quieren aprender, [13].

En las figuras **2** y **3** se muestran los gráficos con las tendencias actuales de Desarrollo y los lenguajes que están ingresando al mercado.

Figura 1: Clase Groovy. Las clases de Groovy permiten hacer uso de características Java que a su vez son nativas en Groov [11]

```
class User
{
    Long id
    Long version

    String email
    String password

    String toString()
        { "$email" }

    def constraints =
    {
        email(email:true)
        password(blank:false, password:true)
    }
}
```

Figura 2: Tendencias de desarrollo [12]

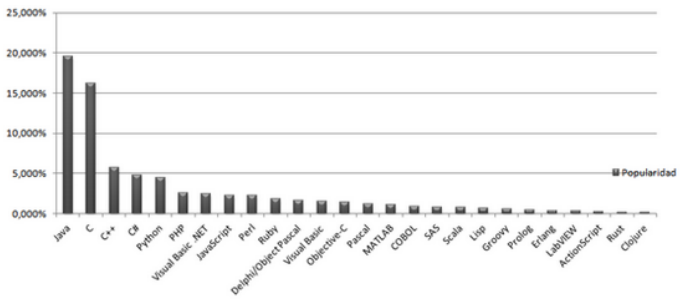
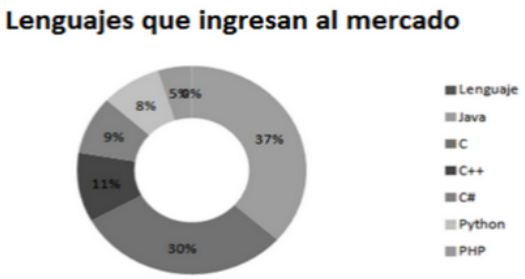


Figura 3: Lenguajes que están ingresando al mercado [13]



En pocas palabras, con Groovy se requieren menos líneas de código, es más legible, el tipado es dinámico, las colecciones son de rápido y fácil acceso, además de que el manejo de las meta-clases facilita manipular o extender un comportamiento existente. En la actualidad existen blogs con los aportes realizados por miembros de la comunidad de desarrolladores Android, en los que se describe cómo desarrollar proyectos móviles en Android, usando Groovy como lenguaje base, mediante tutoriales y plugins para este fin. Algunas de las entradas en las que se puede evidenciar estos casos son: *Developing a native Android App using Groovy, [8] Groovy language support for Android, [14]. Creating Android Apps with Groovy 2.4*, [15]. En estos sitios se pueden ver los pasos que se deben seguir para crear un proyecto Android usando Groovy como lenguaje alterno a Java; así como también se pueden encontrar otros aportes de esta misma índole, usando otros lenguajes.

Existen varias formas de medir el tiempo de ejecución de las aplicaciones, ya sea usando las ecuaciones de coste de tiempo computacional (1) o métodos más simples como restar los tiempos de fin e inicio de un proceso, usando librerías externas que realizan el cálculo o el uso de herramientas de desarrollo, [16]. Este último es el que se utilizó en la prueba presentada en este artículo, debido a que el log del entorno facilita la tarea.

$$T = C \cdot n^2 \tag{1}$$

Ecuación 1. Calculo de tiempo de un algoritmo, donde se C es la constante multiplicativa y n el tamaño de los datos de entrada [17].

4. Materiales y métodos

Para explicar cómo se integra Groovy en Android, se desarrolló la siguiente metodología. Se codifica una aplicación móvil como ejemplo. Luego, como entorno de desarrollo se usa la herramienta Android Studio bajo el API de Google N° 19 y la versión 2.4.6 de Groovy. Esta aplicación se ejecuta sobre el AVD (Android Virtual Device) configurado previamente [18].

Luego, se hace uso de una herramienta, que ya está incluida en el API de Android, llamada Gradle, la cual sirve para automatizar la construcción de los proyectos; es muy flexible para la configuración, pero además ya tiene armadas las tareas por defecto para la mayoría de los proyectos. Esta herramienta es necesaria ya que facilita la implementación y ahorra tiempo al momento de desarrollar, manejando compilaciones incrementales, lo que quiere decir que verifica si hubo algún cambio en el código fuente después de la última compilación, evitando que compile fuentes innecesariamente. Gradle funciona con un lenguaje llamado DSL (Domain Specific Lenguaje) o lenguaje específico de dominio, el cual está escrito en Groovy, por lo que la integración del mismo en Android es bastante sencilla sin necesidad de adicionar librerías [19].

La aplicación consta de una lista en memoria que almacena los nombres temporalmente, un campo de texto para ingresar el nombre de la persona el cual servirá para guardar y filtrar, y una lista en otra vista para mostrar información de la persona sobre la cual se realizará la búsqueda. En las figuras **4** y **5** se muestran las clases Java y Groovy.

Figura 4: Clase en Groovy [20]

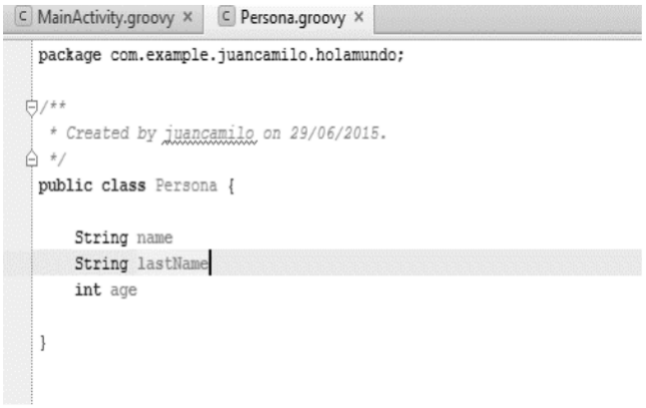
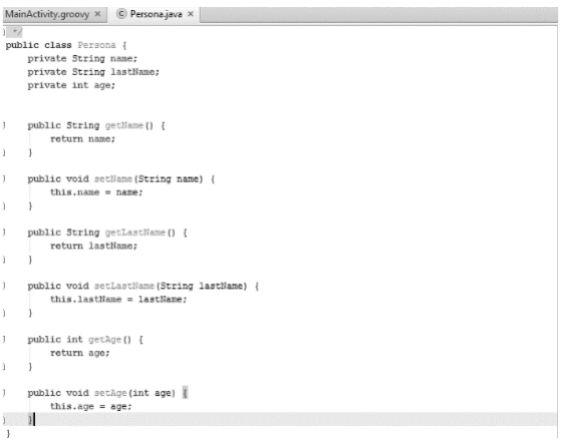


Figura 5: Clase en Java [20]



La prueba consiste en realizar la inserción de un nombre que se almacena en una lista, después se adiciona la funcionalidad de consulta de los nombres almacenados

y posteriormente se adiciona la posibilidad de aplicar un filtro a esa lista, para finalmente poder realizar una comparativa de Groovy contra Java.

Como se puede apreciar, la clase Groovy, a diferencia de la clase Java, no requiere de terminador de línea (;), ni los métodos de encapsulamiento, ya que esto lo hace el lenguaje de manera automática y transparente.

En la figura **6** se puede apreciar la declaración de la lista en Java y el método de creación y búsqueda en ella.

Figura 6: Búsqueda y creación de nombres en una lista en Java [20]

```
    public void onClick(View v){
        String name = ((EditText) findViewById(R.id.name)).getText().toString();
        names.add(name);
        String data[] = null;
        if(names!=null && names.size()>0){
            data = new String[names.size()];
            for(int i=0; i<data.length;i++){
                data[i] = names.get(i);
            }
        }
        ((EditText) findViewById(R.id.name)).getText("");
        abrir(data);
    }
});

Button searchButton = (Button)findViewById(R.id.searchBtn);
searchButton.setOnClick(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        filter = new ArrayList<String>();
        String name = ((EditText) findViewById(R.id.name)).getText().toString();
        for(int i=0; i<names.size();i++){
            if(names.get(i).contains(name)){
                filter.add(name);
            }
        }
        String data[] = null;
        if(filter!=null && filter.size()>0){
            data = new String[filter.size()];
            for(int i=0; i<data.length;i++){
                data[i] = filter.get(i);
            }
        }
        ((EditText) findViewById(R.id.name)).getText("");
        abrir(data);
    }
}

public void abrir(String[] data){
    Intent intent = new Intent(this.List.class);
    intent.putExtra("names",data);
    sartActivity(intent);
}
```

Figura 7: Búsqueda y creación de un elemento de la lista en Groovy [20]

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    names = []
    Button addButton = (Button)findViewById(R.id.button)
    addButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String name = ((EditText) findViewById(R.id.name)).getText().toString()
            names.add(name)
            ((EditText) findViewById(R.id.name)).setText("")
            abrir(names)
        }
    })
    Button searchButton = (Button)findViewById(R.id.search_btn)
    searchButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            filter=new ArrayList<String>()
            String name = ((EditText) findViewById(R.id.name)).getText().toString()
            names.each{it->
                if(it.contains(name)){
                    filter.add(it)
                }
            }
            ((EditText) findViewById(R.id.name)).setText("")
            abrir(filter)
        }
    })
}

public void abrir(data){
    Intent intent = new Intent(this,LIst.class)
    intent.putExtra("names", data)
    startActivity(intent)
}
```

La Figura **7** muestra el mismo proceso de la imagen anterior pero escrito en Groovy. Se puede evidenciar la reducción de código, como también que el proceso de la búsqueda no requiere de ciclos y validaciones debido a que Groovy cuenta con closures (función anidada pero no declarada) para realizar la búsqueda y cuenta con variables por defecto para recorrer la lista, [21].

Para analizar los resultados, se observan las diferencias de tiempo de los lenguajes dinámicos.

5. Resultados

Como indica la tabla **1**, el uso de lenguajes dinámicos reduce el tiempo de desarrollo y construcción aproximadamente en un 33 % en comparación con los lenguajes estáticos. Esta reducción se debe precisamente al hecho de ejecutar búsquedas sin indexar. Aunque esta es una aplicación sencilla que no consume recursos, al momento de realizar aplicaciones a gran escala esto puede ser significativo, como se aprecia a continuación.

Tabla 1: Comparación tiempos de construcción Java – Groovy [20]

Id Proceso	Java	Groovy	Observación
P1	BUILD SUCCESSFUL Total time: 22.756 secs	BUILD SUCCESSFUL Total Time: 18.385 secs	Primera compilacion en la cual se crea la aplicación para la captura de datos
P2	BUILD SUCCESSFUL Total time: 37.225 secs	BUILD SUCCESSFUL Total time: 22.136 secs	En esta compilacion se adiciona la funcionalidad de listar
P3	BUILD SUCCESSFUL Total time: 47.333 secs	BUILD SUCCESSFUL Total time: 29.258secs	Se agrega un filtro para realizar busquedas

En la figuras **8** y **9** se evidencia el comportamiento de los tiempos de las pruebas realizadas en función del tiempo.

Figura 8: Gráfica de comparación de tiempos de ejecución y construcción Java – Groovy [20]

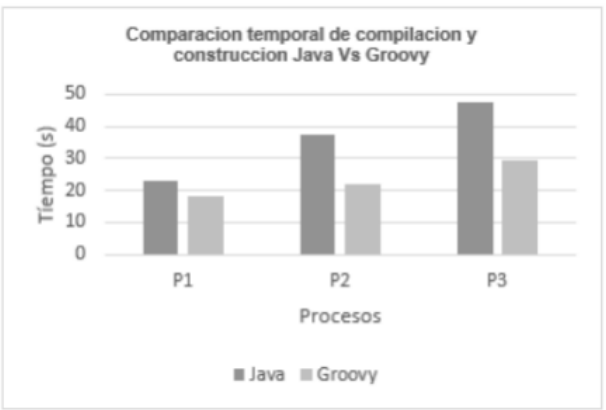


Figura 9: Traza de tiempo generada por la consola de Android Studio [20]

```
Groovy:
02-29 22:15:06.507 18779-18779/org.test2 W/IInputConnectionWrapper: beginBatchEdit on inactive InputConnection
02-29 22:15:06.507 18779-18779/org.test2 W/IInputConnectionWrapper: endBatchEdit on inactive InputConnection
02-29 22:15:06.537 18779-18779/org.test2 W/IInputConnectionWrapper: getExtractedText on inactive InputConnection
02-29 22:15:06.617 18779-18779/org.test2 W/IInputConnectionWrapper: getTextBeforeCursor on inactive InputConnection
02-29 22:15:06.627 18779-18779/org.test2 W/IInputConnectionWrapper: getTextAfterCursor on inactive InputConnection
02-29 22:15:06.627 18779-18779/org.test2 W/IInputConnectionWrapper: getSelectedText on inactive InputConnection
02-29 22:15:06.657 18779-18779/org.test2 W/IInputConnectionWrapper: beginBatchEdit on inactive InputConnection

Java:
02-29 22:18:42.927 18779-18779/org.test2 W/IInputConnectionWrapper: beginBatchEdit on inactive InputConnection
02-29 22:18:42.927 18779-18779/org.test2 W/IInputConnectionWrapper: endBatchEdit on inactive InputConnection
02-29 22:18:42.937 18779-18779/org.test2 W/IInputConnectionWrapper: getExtractedText on inactive InputConnection
02-29 22:18:42.987 18779-18779/org.test2 W/IInputConnectionWrapper: getTextBeforeCursor on inactive InputConnection
02-29 22:18:42.987 18779-18779/org.test2 W/IInputConnectionWrapper: getTextAfterCursor on inactive InputConnection
02-29 22:18:42.987 18779-18779/org.test2 W/IInputConnectionWrapper: getSelectedText on inactive InputConnection
02-29 22:18:43.127 18779-18779/org.test2 W/IInputConnectionWrapper: beginBatchEdit on inactive InputConnection
```

A continuación se muestran los resultados obtenidos de la comparación del tiempo de ejecución entre Java y Groovy. En ellos se puede observar una disminución de 35.82 % como se evidencia en la figura **10**.

Figura 10: Cálculo del porcentaje de disminución de tiempo entre Groovy y Java [20]

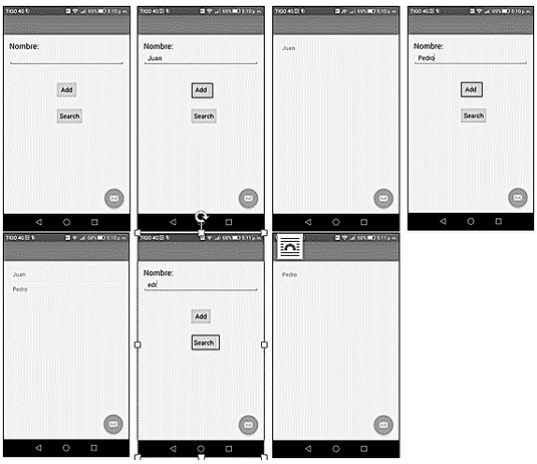
Groovy: Inicio: 22:15:06.507 Final: 22:15:06.657 ➔Final-Inicio = 120 ms

Java: Inicio: 22:18:42.927 Final: 22:18:43.127 ➔Final-Inicio = 187 ms

$$\frac{120 * 100}{187} = 64.17\%$$

100%-64.17% = 35.82%

Figura 11: Aplicación resultante en funcionamiento [20]



Al probar la aplicación se puede evidenciar que la funcionalidad de la aplicación es la misma en ambos casos, aunque para Groovy es más óptimo, sin demeritar la eficiencia de Java, la cual requiere que se implementen funciones, métodos que en Groovy ya vienen integrados al lenguaje por lo cual no es necesario hacer uso de librerías externas. En la figura **11** se puede observar el resultado de las aplicaciones en funcionamiento paso a paso.

6. Conclusiones

Los lenguajes dinámicos permiten crear aplicaciones de manera ágil y de fácil sostenimiento debido a la naturalidad de su código, ya que es más flexible y permite modificarse de manera más sencilla; por tanto, al hacer uso de lenguajes dinámicos se disminuyen los tiempos de la creación de código, debido a la integración de funciones y librerías Java, así como la facilidad para integrarse con Spring. Estos lenguajes facilitan el mantenimiento y la realización de pruebas ya que es soportada por varios IDE'S que facilitan estas tareas; con estos se pueden elaborar desarrollos completos en muy pocas líneas comparados con otros lenguajes, ya que Groovy permite hacer uso de clases y librerías Java para integrarlas como parte de sí en el desarrollo de una aplicación.

Por otro lado, facilitan la corrección de errores ya que nos permite modificar las variables y objetos en tiempos de ejecución, [21], Groovy permite al desarrollador adaptar una clase simple Java a la sintaxis Groovy lo cual facilita que los desarrolladores Java se adapten con mayor facilidad.

Con el desarrollo de la aplicación prueba, se pude evidenciar, que haciendo uso de Groovy en Android obtenemos un 32.64 % de reducción de tiempo en desarrollo y construcción del paquete Apk, lo cual es en gran medida, una ventaja al momento de desarrollar software.

Referencias

[1] N. Páez, "Construcción de software: una mirada ágil" . EDUNTREF, 2014.

[2] E. Guerrero, "Analisis y diseño de software". 2012 [En línea]. Disponible en: http://wwwefrainguerrero.blogspot.com/2012/04/ciclo-de-vida-scrum.html

[3] C. M. Gabriel, "Tecnologías Móviles - Trabajo final_11.pdf". 2006. [En línea]. Disponible en: https://www.google.com.co/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CB8QFjAAahUKEwj0ucSn-urGAhULqR4KHT9vDWY&url=http%3A%2F%2Fwww.i.edu.mx%2Faportaciones%2Ftrabajo%2520final_11.pdf&ei=rpCtVfSqOIvSer_etbAG&usg=AFQjCNEhC6BEYqHnosLFeV7tQtSy

[4] A. N. Gonzalez, "Xatakandroid". 2011. [En línea]. Disponible en: http://www.xatakandroid.com/sistema-operativo/que-es-android

[5] E. C. Ecured, “ Ecured” [En línea]. Disponible en: http://www.ecured.cu/Multiplataforma#Plataforma_Java

[6] ZeruGiran, ”Tipado estático versus tipado dinámico”2016. [En línea]. Disponible en: http://qbit.com.mx/blog/2012/01/16/tipado-estatico-vs-tipado-dinamico/

[7] TADP, “ Técnicas Avanzadas de Programación” . 2012. [En línea]. Disponible en: https://sites.google.com/site/utntadp/cursadas-anteriores/2012c1-clases/clase-10

[8] Object Partners, “Developing a native Android App using Groovy”. 2014. [En línea]. Disponible en: https://objectpartners.com/2014/09/04/developing-native-android-app-using-groovy/

[9] K. Barclay, “Groovy Programming”. 2007. [En línea]. Disponible en: https://books.google.com.co/books?id=kV8GYTCDf9AC&printsec=frontcover&dq=Groovy+Programming&hl=es&sa=X&ved=0ahUKEwjOt9mD2dDNAhWBcyYKHeUADcUQ6AEIIzAA

[10] Groovy, ”Documentation”. [En línea]. Disponible en: http://www.groovy-lang.org/documentation.html

[11] N. Brito, “ Manual de desarrollo web con GrailsÏmaginaworks Software Facto” , 2009. [En línea]. Disponible en: https://books.google.com.co/books?id=Wa9AXT7zHVwC&pg=PA19&lpg=PA19&dq=manual+de+desarrollo+web+con+grails+nacho+brito&source=bl&ots=YIgM4kkV8r&sig=JXRK2V1b2vra8D_JgXloAB2AkIc&hl=es&sa=X&ved=0ahUKEwjJ6MbqorfKAhVDXB4KHV8mDScQ6AEIGjAA#v=onepage&q=manual%20de

[12] Tiobe Software, ”TIOBE index for October 2015”. Octubre 2015. [En línea]. Disponible en: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[13] T. Rodriguez, “ genbetadev” Agosto 2014. [En línea]. Disponible en: http://www.genbetadev.com/herramientas/diez-tecnologias-que-los-javeros-amamos-o-al-menos-hablamos-bien-de-ellas

[14] A. Reitz, ”GitHub”[En línea]. Disponible en: https://github.com/groovy/groovy-android-gradle-plugin

[15] M. Scharhag, “javacodegeeks.com”. Febrero 2015. [En línea]. Disponible en: http://www.javacodegeeks.com/2015/02/creating-android-apps-groovy-2-4.html

[16] Deerme, “Deerme”. 1 Agosto 2010. [En línea]. Disponible en: http://deerme.org/java/medir-tiempo-de-ejecucion-en-java

[17] M. Mascaró Portells, P. A. Palmer Rodríguez y A. Jaume Capó, “Análisis del coste computacional”. [En línea]. Disponible en: http://dmi.uib.es/ mascport/tp/perweb/tema1.html

[18] Android Developers, “Managing AVDs with AVD Manager”. [En línea]. Disponible en: http://developer.android.com/tools/devices/managing-avds.html#skins

[19] Hosain.net, “Getting Started With Android Development Using Groovy 2.4 and Android Studio”. 7 Febrero 2015. [En línea]. Disponible en: http://hosain.net/2015/02/07/getting-started-with-android-development-using-groovy-2.4-and-android-studio.html

[20] J. C. Sosa Suarez, “ Ejercicio práctico de la implementacion Android Groovy” , Tesis para optar por el título de ingeniería telemática, Universidad distrital Francisco José de Caldas, Bogotá, Colombia, 2015.

[21] H. L. Weissmann, “Groovy vs Java: listas”. Septiembre 2011. [En línea]. Disponible en: http://www.itexto.net/devkico/?p=976