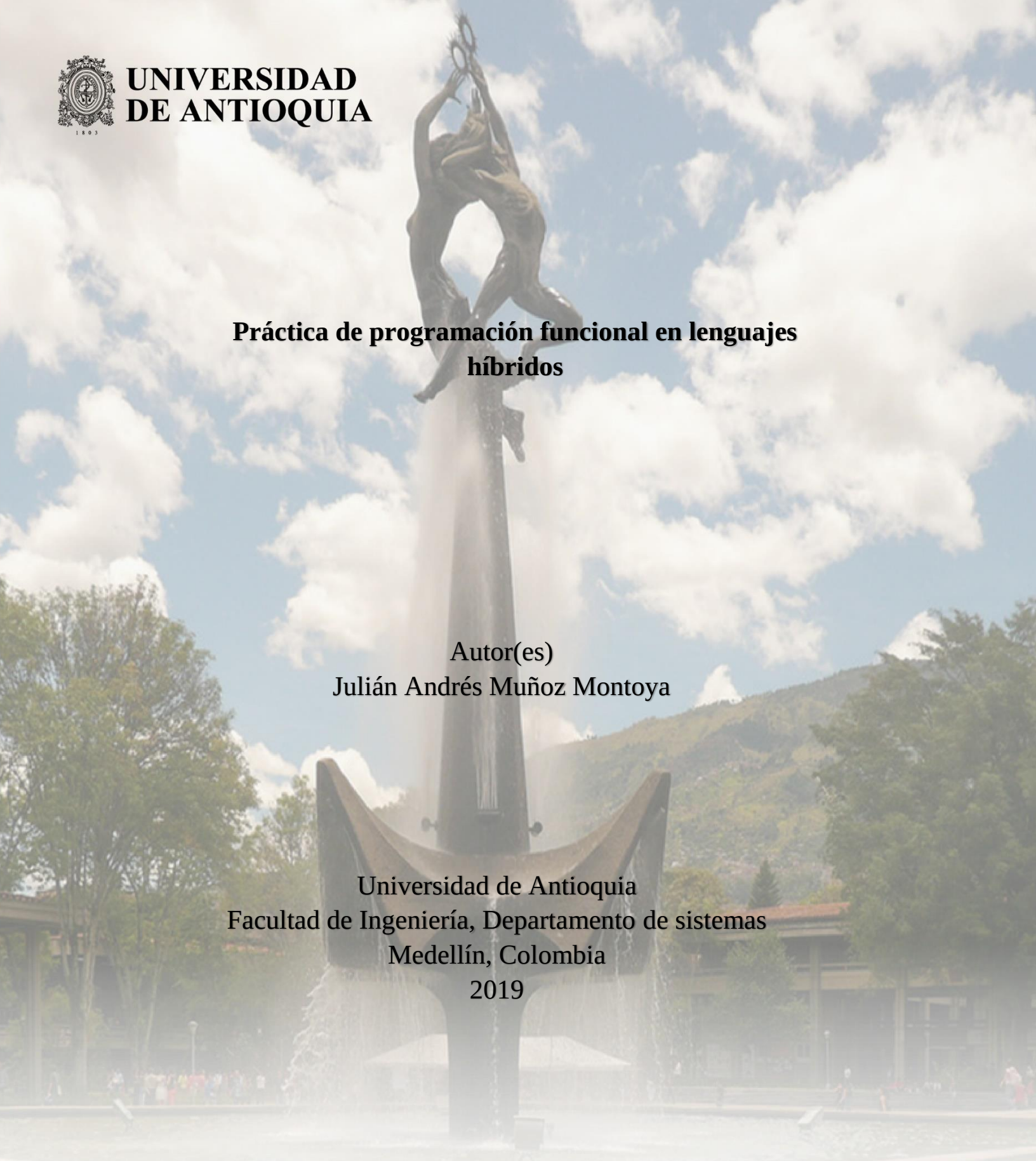

UNIVERSIDAD DE ANTIOQUIA

Práctica de programación funcional en lenguajes híbridos

Autor(es)
Julián Andrés Muñoz Montoya

Universidad de Antioquia
Facultad de Ingeniería, Departamento de sistemas
Medellín, Colombia
2019

Práctica de programación funcional en lenguajes híbridos

Julián Andrés Muñoz Montoya

Informe final de prácticas
como requisito para optar al título de:
Ingeniero de sistemas

Asesor interno
Sandra Patricia Zabala Orrego

Asesor externo
David Quinto Pandales

Julián Andrés Muñoz Montoya Ingeniero de sistemas

Universidad de Antioquia
Facultad de Ingeniería, Departamento de sistemas
Medellín, Colombia
2019

Tabla de Contenido

Tabla de figuras

1. Resumen

En esta práctica empresarial fue asignado al estudiante la tarea de migrar un sistema de turnos. La intención de esta migración era mejorar la mantenibilidad del sistema por medio de la programación funcional y el lenguaje Kotlin. El estudiante es ingresado a un equipo SCRUM, el cual trabajó bajo la tutorial del líder técnico de la empresa.

Durante la práctica el equipo de trabajo tenía como otro objetivo principal, identificar técnicas de programación funcional. Estas técnicas fueron las funciones puras, las funciones de alto orden, la inmutabilidad y las mónadas.

Al finalizar el proyecto el estudiante entrega un sistema de turnos básico con 3 módulos, el de configuración, el transaccional y el de reportes; además de esto se suma el aprendizaje practicante sobre programación funcional y desarrollo de sistemas mantenibles y escalables. Este conocimiento se suma como un insumo más para la empresa, pues este conocimiento será utilizado para desarrollos futuros.

2. Introducción

La empresa grupo ESOFT ha venido migrando todos sus productos comerciales a la programación funcional, con el objetivo de crear programas más mantenibles y siguiendo la actual tendencia mundial de integrar está a productos comerciales (Coens, 2015), para así modernizar sus productos comerciales.

En la última década la programación funcional se ha utilizado en grandes compañías como Facebook (Coens, 2015), Twitter (Corneillet, 2018) y Disney (Roberts, 2018), debido a la facilidad que provee para construir software robusto, complejo y escalable. La programación funcional ha estado desde hace varios años, sin embargo, paradigmas más sencillos como la programación imperativa y la programación orientada a objetos, mantuvieron a la programación funcional en la sombra por muchos años.

Al día de hoy los retos en la industria de software se han vuelto más complejos, por ejemplo, la concurrencia y la falta de control sobre el estado de las aplicaciones, pueden ocasionar fallos difíciles en sistemas imperativos u orientados a objetos, estas condiciones adversas pueden ser evitadas más fácilmente con la programación funcional (Hoyos y Puertas, 2017).

La tarea del practicante era participar en la migración del sistema de turnos, pues el sistema anterior ya tenía muchos años en operación. La modernización del sistema tenía como objetivo identificar técnicas y buenas prácticas de programación funcional. El trabajo del estudiante se limitaba a migrar el sistema de turnos solamente en el lado servidor, manteniendo la compatibilidad con los sistemas clientes antiguos del sistema de turnos.

El desarrollo de este sistema demuestra que la programación funcional puede ser utilizada en la industria comercial del software a pequeña y mediana escala.

3. Objetivos

Identificar qué elementos y técnicas de la programación funcional pueden ser incorporados como buenas prácticas en el proceso de desarrollo de la compañía.

3.1 Objetivos Específicos:

3.1.1 Implementar tareas cotidianas de programación funcional durante la práctica.

3.1.2 Identificar y comparar buenas prácticas de programación funcional según el número de tareas implementadas semanalmente, para incorporarlas a productos comerciales.

4. Marco teórico

La programación funcional consiste en componer programas a partir de funciones, estas funciones reciben unos parámetros retornan un valor o bien una función. Las funciones que retorna otra función, se llaman funciones de alto orden, se explicarán en esta sección. A continuación, se explican cada una de las técnicas de programación funcional seleccionadas.

4.1. Funciones puras:

Son funciones cuya invocación con el mismo parámetro, siempre retorna el mismo resultado a lo largo del tiempo, por lo tanto, el resultado de una función pura siempre es predecible (Chiusano y Bjarnason, 2014). En la figura 1 se puede ver un ejemplo de una función pura.

```
val duplicar: (Int) -> Int = (x) -> x * 2
```

Figura 1. Función pura para duplicar un número en Kotlin.

El tipo de la función de la figura 1 precedido por los dos puntos indica que recibe un parámetro de tipo entero y devuelve un resultado de tipo entero.

Por el contrario, las funciones impuras generan resultados impredecibles, ya que estos si cambian a lo largo de tiempo, véase un ejemplo en la figura 2.

```
var x = 1
...
fun sumaImpura(y: Int): Int {
    x = y + x
    return x
}
```

Figura 2. Función impura.

En la función de la figura 2 se ve como se aumenta el valor de *x* y luego se retorna cada vez que se ejecuta la función, por lo cual el resultado no puede ser predecible.

4.2. Funciones de alto orden:

Las funciones de alto orden son funciones que reciben otras funciones como parámetro (Chiusano y Bjarnason, 2014), un ejemplo es una función que calcula el resultado de una operación entre 2 parámetros numéricos, como la función *calcular* de la figura 3.

```
fun calcular(x: Int, y: Int, f: (Int, Int) -> Int) = f(x, y)

val a = calcular(2, 5, (x, y) -> x * y)
```

Figura 3. Una función de alto orden.

Las funciones de alto orden más comunes en la programación funcional son map filter y reduce, las cuales se aplican a las listas.

4.1.1. Map:

Aplica una función de transformación a cada elemento de la lista (Kurt, 2018). En la figura 4 podemos ver un ejemplo del sistema de turnos, se tiene una lista de atenciones, donde uno de sus campos es el tiempo que tuvo que esperar un cliente en la cola, en este caso transformamos cada atención en su tiempo en cola, ver figura 4.

```
val tiemposEnCola = turnos.map { t -> t.tiempoEnCola }
```

Figura 4. Tiempos en cola de una lista de atenciones.

4.1.2. Filter:

Esta función filtra los elementos de una lista que no cumplan con el predicado pasado como parámetro (Kurt, 2018). Un ejemplo en el sistema de turnos sería filtrar los turnos que no sean para atención prioritaria en un hospital, ver figura 5. Nótese que el predicado pasado como parámetro, retorna un número booleano, solo los turnos que retornan verdadero pueden ser agregados al resultado final.

```
val turnosAtencionPrioritaria =
    turnos.filter { t -> t.fila == TipoFila.PRIORITARIA }
```

Figura 5. Turnos para atención prioritaria en un hospital.

4.1.3. Reduce:

Reduce todos los elementos a un solo valor, esta función recibe como parámetro una función de combinación (Hughes, 1989). En el sistema de turnos podemos calcular el tiempo total en cola de todos los turnos, simplemente sumándolos todos con la función reduce, ver figura 6.

```
val tiempoEnColaTotal = turnos.reduce { (tx, ty) ->
    tx.tiempoEnCola + ty.tiempoEnCola
}
```

Figura 6. Tiempo total en cola para todos los turnos.

4.3. Mónada:

Una mónada es un contenedor que permite abstraer una computación de los efectos adversos que esta puede generar o estar sometida, permitiendo devolver este efecto adverso como un valor o reemplazarlo por otro valor. Los efectos adversos pueden ser errores, un número no predecible de valores, un valor que puede estar ausente o un valor que será calculado en un futuro (Wadler, 1992).

Cada efecto adverso se define como tipo de datos genérico, por ejemplo, la mónada Either que fue utilizada en nuestro sistema, esta mónada tiene como propósito, retornar un valor de tipo derecho cuando la computación ha sido exitosa y un valor de tipo izquierdo cuando la computación ha fallado.

En nuestro sistema de turnos, la generación de un reporte de las atenciones prestadas en el día puede retornar una respuesta HTTP exitosa de tipo derecho o una respuesta de error de tipo izquierdo HTTP fallida, ver ejemplo en la figura 7.

```
fun extraerReporte(): Either<Error, Success> = generarReporte()
    .map { reporte ->
        when {
            reporte.isEmpty() ->
                Error("No se han atendido clientes todavia.")
            else -> Success(reporte)
        }
    }
```

Figura 7. División de enteros usando el tipo Either.

5. Metodología

En el comienzo de la práctica se recibió capacitación en el lenguaje Kotlin y los fundamentos de la programación funcional. Durante el mes de febrero y la primera semana de marzo se inició con la lectura de documentación y con retos de programación funcional.

Entre los meses de marzo y la primera semana de mayo se realizó el desarrollo del nuevo sistema de turnos bajo la metodología SCRUM con sprints de 2 semanas cada uno, con un equipo de 4 personas donde ocupe el rol de desarrollador, bajo la tutoría del asesor externo quien cumple el rol de scrum máster.

A continuación, se describe cada sprint.

5.1. Sprint 0 - Módulo de configuración:

Este módulo fue desarrollado utilizando la función de alto orden *filter* para evitar información incorrecta que pudiera dañar veracidad de información almacenada, luego con la función map se realizaban las transformaciones necesarias para.

5.2. Sprint 1 - Módulo transaccional:

Este módulo fue el más difícil, ya que al principio se utilizó mucho la mutabilidad y los cambios de estado en el manejo de las colas de espera, por ser lo más sencillo de implementar, pero estos cambios de estado estaban generando bugs, muchas veces difíciles de rastrear o predecir, por lo cual se optó finalmente por la imputabilidad a costo de un mayor uso de memoria, para garantizar una mayor fiabilidad del sistema.

5.3. Sprint 2 - Módulo de reportes - Atenciones consolidado:

En este primer reporte se implementó con las funciones de alto orden map, filter, reduce y zip, esta última nos permitió combinar varias listas para crear reportes

informativos y fáciles de entender. En este reporte se tuvieron dificultades para el manejo y la presentación de errores al usuario.

5.4. Sprint 2 - Módulo de reportes – Servicios por fecha:

En este reporte se solucionaron los problemas en el manejo de errores, presentes en el reporte anterior, con la ayuda de la mónada Either, permitiéndonos retornar una excepción como tipo izquierdo o el reporte como tipo derecho, de una manera limpia, evitando la complejidad de los bloques try catch tradicionales de java.

Al principio de cada sprint se realizaba una reunión de planning donde se dividía cada problema en tickets o tareas, que luego eran puestas en el backlog y cada desarrollador se iba asignado tareas durante el desarrollo.

Al final de cada día se desplegaba el proyecto y al inicio de cada día se realizaban pruebas al proyecto y cada bug encontrado era reportado en el backlog del proyecto, durante el día se realizaban correcciones y se realizaban más funcionalidades, para luego desplegar una nueva versión al final del día para continuar con el ciclo. Las pruebas diarias se realizaron utilizando los sistemas clientes y pruebas automáticas con Junit.

La comunicación del equipo se daba mayormente de manera personal, ya que la oficina es un espacio abierto y la idea era solucionar cualquier duda, problema o inquietud inmediatamente.

El proyecto se alargó 2 semanas más, en las cuales se verificó completamente el funcionamiento del sistema y se integró con los sistemas clientes.

A mediados de mayo con 2 semanas de retraso en el calendario, se empezó con la recolección de las lecciones aprendidas. Durante este periodo se hizo una retroalimentación con las nuevas técnicas aprendidas de programación funcional, se recopilaron errores tanto en desarrollo como en comunicación y se realizaron jornadas de aprendizaje sobre programación funcional.

6. Resultados y análisis

Al final del desarrollo adquirí fluidez en el lenguaje Kotlin, siendo productivo para desarrollar software utilizando técnicas de programación funcional.

En cuanto al sistema de turnos de la empresa, se ha obtenido un sistema moderno y extensible para el futuro, los entregables se describen a continuación:

6.1. Módulo de configuración:

Permite consultar, almacenar, editar y eliminar la información que alimenta el sistema de turno. La información son las sedes en las que una empresa provee servicios, los tipos de servicio que presta una organización y información sobre los asesores que prestan los servicios.

6.3. Módulo transaccional:

Se encarga de recolectar y procesar la información del sistema de turnos durante su funcionamiento. Los procesos que se ejecutan en este módulo son el manejo del estado para las colas de atención, autenticar los asesores en los puestos de atención, generar turnos y almacenar el resumen de una atención, en la cual se prestan uno o más servicios a un cliente.

6.4. Módulo de reportes:

Este módulo se encarga resumir la información de todas la atenciones y servicios prestados por la entidad cliente de nuestro sistema, a un administrador del sistema, para que este pueda conocer la calidad del servicio, la disponibilidad, la rapidez y la demanda en su entidad, para que así este pueda tomar decisiones para mejorar la oferta de productos y la calidad del servicio con el objetivo de generar valor para la entidad.

Los reportes que se realizaron son los siguientes:

6.4.1. Servicios por fecha:

Este reporte es un resumen histórico de cada servicio que puede ser prestado en la entidad. Este reporte muestra en cada el resumen de cada servicio.

6.4.2. Atenciones consolidado:

Este reporte es un histórico de atenciones prestadas durante un periodo de tiempo específico, en este se hace un resumen de cada atención prestada.

En la figura 8 se puede ver el resultado del reporte de servicios por fecha, el cual retorna del lado servidor, un objeto json, a este reporte se le pasaba como parámetros la fecha de inicio y la fecha fin en la que se generarán los reportes, por cada día se genera un reporte. En caso de fallo se retornaba un Either con una respuesta de error, el reporte era un Either de tipo derecho lista de reportes.

La imagen es de un reporte de prueba generado para una reunión de resultados durante la etapa de lecciones aprendidas, este reporte puede ser visto en la figura 8.

Figura 8. Reporte de servicios por fecha.

En la evaluación de rendimiento final, se le entregó al estudiante una tarea con un nivel de complejidad similar al de la prueba de selección, los resultados de esta prueba y de las pruebas de rendimiento realizadas durante la práctica, se pueden ver en la figura 9.

En los resultados se puede apreciar la mejora del estudiante gracias a la destreza adquirida durante la práctica, se observa una mejora en el uso de programación funcional en java, teniendo un mejor resultado que usando Kotlin de manera imperativa.

El bajo rendimiento en usando programación imperativa en java, se debe al desuso de este lenguaje de manera imperativa y al estar cada vez más acostumbrado al lenguaje Kotlin. La programación funcional en java mejora porque en encadenamiento de funciones de alto orden en java y Kotlin son bastantes similares.

El rendimiento laboral se mide, por el número de tareas de 10 puntos realizadas semanalmente, estas tareas eran la gran mayoría. En nuestro proyecto de turnos, 10 puntos equivalen a tareas que deberían demorar no más de 8 horas. El gráfico de rendimiento puede verse en la figura 10.

Se tuvo un bajo rendimiento en las primeras semanas debido por la etapa de formación, aprendiendo Kotlin y programación funcional, luego se observa cómo se produce un acoplamiento al nivel del equipo en las 3 últimas semanas.

Nótese las 2 semanas extras de retraso en las que se realizaron modificaciones significativas en el sistema.

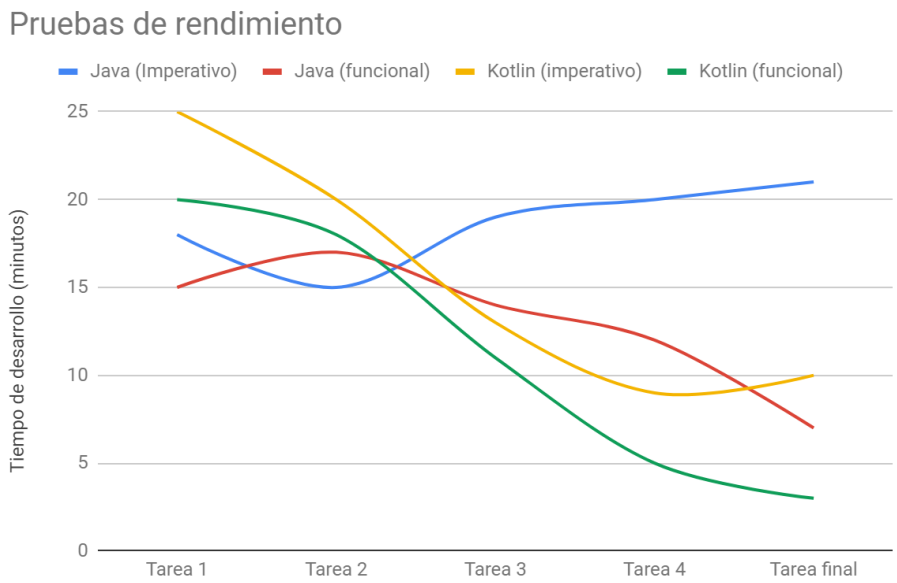

Figura 9. Resultados de las pruebas de rendimiento.

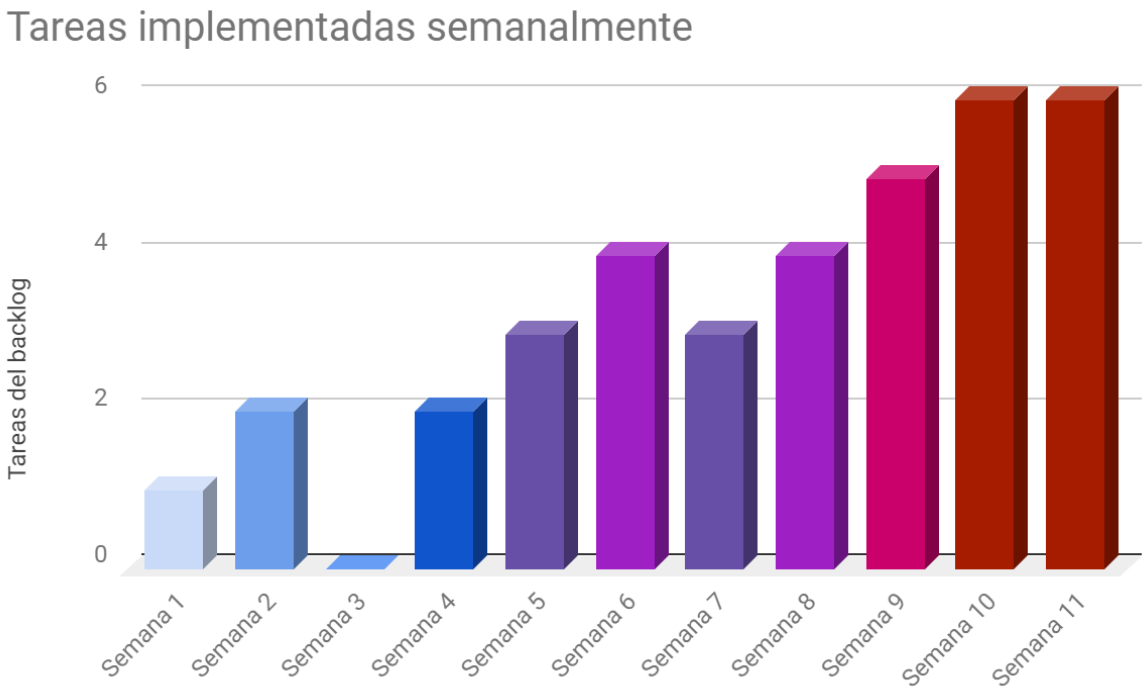

Figura 10. Número de tareas implementadas semanalmente.

7. Conclusiones

7.4. La programación funcional puede tardar tiempo en ser asimilada, pero una vez superado este obstáculo, un desarrollador puede volverse más productivo ya que reduce la cantidad de errores en su código debido a la mutación de estados y es capaz de crear código más mantenible y escalable (Hu, Hughes y Wang, 2015).

7.5. Hoy en día cada vez más lenguajes comerciales como Kotlin, Python y Swift; Adoptan la programación funcional de manera parcial, lenguajes como Scala experimentan una demanda creciente; finalmente Haskell que apareció en los noventas ha despertado interés como lenguaje básico para aprender programación funcional.

7.6. Los paradigmas orientados a objetos e imperativo combinados con la programación funcional le permiten a un desarrollador ser productivo, evitar errores como punteros nulos y cambios de estado, ser versátil y crear software escalable.

7.7. La comunicación efectiva en un equipo me permitió superar las dificultades de aprender la programación funcional y a mejorar mis habilidades en los paradigmas orientados a objetos e imperativo.

7.8. Actualmente con las altas exigencias del mundo moderno, la programación funcional será más relevante y necesaria para el desarrollo de sistemas complejos y de gran escala; saliendo así del desuso que ha sufrido desde hace 5 décadas.

8. Referencias

8.1. Chiusano, P. y Bjarnason, R. (Primera edición). (2014). *Functional programming in scala. Shelter Island*, EE. UU.: Manning.

8.2. Coens, J. (Erlang Solutions). (2015). *The road to running haskell at Facebook scale*. [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=sl2zo7tzrO8&t=1s

8.3. Corneillet, I. (FunctionalTV). (Diciembre del 2018). *How Twitter teaches Scala*. [Archivo de video].
Recuperado de https://www.youtube.com/watch?v=nzGl7laiz6g&t=1s

8.4. Hoyos, J. y Puertas, A. (Octubre del 2017). Desempeño de los Paradigmas Funcional y Orientado a Objetos en Arquitecturas Multicore: Caso de Estudio Haskell-Java. *Inf. tecnología La Serena*. *28*(5).

8.5. Hu, Z., Hughes, J. y Wang, M. (Septiembre del 2015) How functional programming mattered. *National Science Review*, vol 2, issue 3. Septiembre 2015. Págs 349–370.

8.6. Hughes, J. Why functional programming matters. (Abril del 1989). *The computer journal*, *32*(2). Págs 98 - 101.

8.7. Kurt, W. (2018). *Get programming with Haskell*. Shelter Island, EE. UU.: Manning.
8.8. Roberts, M. (Skills Matters). (Diciembre del 2018). *Type-Driven Development in Practice: Cats and Akka HTTP. SCALA EXCHANGE*. Recuperado de https://skillsmatter.com/skillscasts/13003-type-driven-development-in-practice-cats-and-akka-http
8.9. Wadler, P. (Enero del 1992). *19th Annual Symposium on Principles of Programming Languages.* The essence of functional programming. ACM SIGPLAN. Santa Fe, New Mexico, EE. UU.