**m**

**UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO**

CENTRO UNIVERSITARIO UAEMEX ECATEPEC

“Que es la programación”

**Programación Móvil**

Maestría en Ciencias de la Computación

P R E S E N T A

**David Felipe Ortiz Guerrero**

**Logotipo

Descripción generada automáticamente**

ECATEPEC DE MORELOS, ESTADO DE MÉXICO Agosto/2024

Contenido

[Introducción 3](#_Toc174740459)

[Propuesta 3](#_Toc174740460)

[Objetivos 4](#_Toc174740461)

[Marco Teórico 4](#_Toc174740462)

# Introducción

La tecnología móvil ha transformado profundamente la manera en que interactuamos con el mundo, desde la comunicación hasta el entretenimiento y el trabajo, los dispositivos móviles han pasado a ser una parte integral de la vida cotidiana. Como resultado, la demanda de aplicaciones móviles de alta calidad ha aumentado exponencialmente. Hoy en día, existe una aplicación para casi cualquier necesidad, y las empresas buscan continuamente formas de mejorar la experiencia del usuario y optimizar el rendimiento de sus aplicaciones.

Por otra parte, desarrollar aplicaciones móviles presenta desafíos únicos, por ejemplo la fragmentación del mercado móvil es uno de los mayores retos. Hay una gran variedad de dispositivos, sistemas operativos y versiones de software que los desarrolladores deben tener en cuenta, esto implica que una aplicación debe ser compatible con múltiples configuraciones de hardware y software para alcanzar a la mayor cantidad de usuarios posible. Además, las aplicaciones móviles deben ser eficientes en términos de uso de recursos, como batería y memoria, y ofrecer interfaces de usuario intuitivas que funcionen sin problemas en pantallas de diferentes tamaños y resoluciones.

La creciente demanda de aplicaciones móviles ha generado una alta necesidad de desarrolladores especializados en este campo. Las empresas están invirtiendo recursos significativos en el desarrollo de aplicaciones móviles, lo que ha aumentado la presión para crear software que no solo sea funcional, sino también innovador, seguro y fácil de usar, esta demanda ha llevado a la búsqueda de nuevas metodologías y paradigmas de programación que puedan facilitar y optimizar el proceso de desarrollo móvil.

# Propuesta

La propuesta es explorar y evaluar los diferentes paradigmas de programación—descriptiva, declarativa, orientada a objetos y basada en eventos, así comparativamente analizar cómo cada uno de ellos puede abordar los desafíos específicos de la programación móvil. Se sugiere una evaluación comparativa para determinar cuál de estos paradigmas ofrece la mejor solución en términos de eficiencia, escalabilidad, y facilidad de mantenimiento en el desarrollo de aplicaciones móviles.

Esta sugiere que una combinación estratégica de estos paradigmas podría ofrecer una solución más robusta a los desafíos actuales. Por ejemplo, la programación declarativa podría simplificar la creación de interfaces de usuario dinámicas, mientras que la programación orientada a objetos podría ofrecer una estructura sólida y reutilizable para la gestión de datos y lógica de la aplicación.

Para aplicar esta propuesta, se podría desarrollar un prototipo de aplicación móvil que implemente los diferentes paradigmas en diferentes partes de la aplicación. Posteriormente, se llevaría a cabo una evaluación de su desempeño, facilidad de desarrollo y mantenimiento para determinar cuál ofrece las mejores ventajas.

# Objetivos

Objetivo General:

Evaluar el impacto de diferentes paradigmas de programación en la eficiencia y calidad del desarrollo de aplicaciones móviles.

Objetivos Específicos:

1. Analizar la evolución de la programación y su relevancia en el contexto actual del desarrollo de software móvil.
2. Comparar y contrastar los paradigmas de programación descriptiva, declarativa, orientada a objetos y basada en eventos.
3. Proponer un marco de trabajo que combine los mejores aspectos de cada paradigma para el desarrollo eficiente de aplicaciones móviles.
4. Implementar y evaluar el marco propuesto en un proyecto piloto de desarrollo de una aplicación móvil, analizando su desempeño en términos de eficiencia, escalabilidad y facilidad de mantenimiento.

# Marco Teórico

La programación es la práctica de escribir instrucciones detalladas para que una computadora realice tareas específicas, estas instrucciones, o "código", están escritas en un lenguaje de programación que la computadora puede interpretar y ejecutar. A través de la programación, es posible crear desde simples scripts que automatizan tareas repetitivas hasta complejos sistemas operativos y aplicaciones que gestionan vastos recursos y datos.

La evolución de la programación ha sido un viaje desde lo rudimentario hasta lo altamente abstracto y accesible. Los primeros lenguajes de programación, como el ensamblador, estaban directamente relacionados con el hardware y requerían un conocimiento profundo de la arquitectura de la computadora. Estos lenguajes eran extremadamente específicos y difíciles de usar, lo que limitaba su accesibilidad.

Con el tiempo, surgieron lenguajes de alto nivel como Fortran, COBOL y LISP en los años 50 y 60, que abstraían detalles del hardware, permitiendo a los programadores enfocarse más en la lógica de los problemas que estaban resolviendo, estos lenguajes establecieron las bases de la programación moderna al introducir conceptos como variables, bucles y estructuras de control de flujo.

En las décadas siguientes, la programación continuó evolucionando con la introducción de lenguajes más avanzados como C, que ofrecía un equilibrio entre el control del hardware y la abstracción, y C++, que introdujo la programación orientada a objetos. Lenguajes como Java y Python, que surgieron en los 90, llevaron la programación a nuevas alturas con sus capacidades multiplataforma y facilidad de uso.

Hoy en día, la programación sigue evolucionando, con nuevas tendencias como la programación funcional y la programación reactiva ganando popularidad. La programación funcional, que se basa en funciones matemáticas, ha ganado tracción debido a su capacidad para manejar datos inmutables y facilitar la concurrencia, aspectos críticos en el desarrollo de aplicaciones modernas.

Por otro lado, la programación reactiva se enfoca en la propagación automática de cambios y es muy utilizada en el desarrollo de interfaces de usuario y aplicaciones en tiempo real.

Además, la evolución de los entornos de desarrollo integrados (IDEs) y herramientas de automatización ha hecho que el proceso de programación sea más eficiente, permitiendo a los desarrolladores concentrarse en la lógica y la creatividad en lugar de en detalles técnicos.

La programación descriptiva es un paradigma donde el desarrollador se enfoca en describir qué resultado se desea obtener, en lugar de especificar cómo lograrlo, esto significa que el código describe el objetivo final, dejando que el motor de ejecución o compilador determine los pasos necesarios para alcanzar ese objetivo. Este enfoque contrasta con la programación imperativa, donde el desarrollador especifica una secuencia exacta de pasos a seguir.

**Características principales:**

* **Abstracción:** La programación descriptiva abstrae los detalles del control de flujo y la gestión de estados, permitiendo a los desarrolladores centrarse en la lógica del problema.
* **Declaración del objetivo:** En lugar de escribir instrucciones paso a paso, se declara el objetivo final que se quiere alcanzar.
* **Automatización de procesos:** El sistema subyacente decide cómo ejecutar el código para cumplir con la descripción proporcionada.

SQL (Structured Query Language) es un ejemplo clásico de programación descriptiva. Cuando se escribe una consulta SQL, el programador especifica qué datos desea obtener, pero no cómo buscarlos en la base de datos. El motor de base de datos interpreta la consulta y decide la mejor manera de ejecutarla. Por ejemplo:

|  |  |
| --- | --- |
| 1 | SELECT nombre, edad FROM usuarios WHERE ciudad = 'Madrid'; |

En esta consulta, el programador describe qué datos quiere (nombre y edad de los usuarios que viven en Madrid), pero no especifica cómo el motor de base de datos debe recuperar esos datos.

**Ventajas y desventajas:**

* **Ventajas:** La programación descriptiva puede simplificar el desarrollo de software al reducir la cantidad de detalles técnicos que el desarrollador necesita manejar. Esto puede resultar en un código más claro, más fácil de mantener y menos propenso a errores.
* **Desventajas:** Puede ofrecer menos control sobre el rendimiento y la ejecución del código, lo que a veces puede resultar en soluciones menos óptimas en comparación con la programación imperativa.

Por otra parte, la programación declarativa es un paradigma que se centra en describir lo que se desea lograr sin especificar los pasos para lograrlo. A diferencia de la programación imperativa, donde el flujo de control es explícito, en la programación declarativa, el flujo de control es implícito y gestionado por el lenguaje o el motor que ejecuta el código.

**Características principales:**

* **Enfoque en el resultado:** Los desarrolladores describen el estado final deseado, y el lenguaje se encarga de cómo alcanzarlo.
* **Menos control de flujo:** No se requiere que el programador controle explícitamente la secuencia de operaciones.
* **Inmutabilidad:** Muchos lenguajes declarativos favorecen el uso de datos inmutables, lo que puede ayudar a prevenir errores comunes relacionados con la mutación de estado.

Lenguajes como Haskell y Prolog son ejemplos prominentes de programación declarativa. Haskell, un lenguaje puramente funcional, permite a los desarrolladores definir funciones y relaciones matemáticas sin preocuparse por el orden en que se ejecutarán. Prolog, por otro lado, es un lenguaje lógico donde los desarrolladores definen reglas y hechos, y el motor de Prolog resuelve consultas basadas en estos.

En el desarrollo de interfaces de usuario, frameworks como React (JavaScript) emplean un enfoque declarativo. Por ejemplo, en React, los desarrolladores describen cómo debería verse la interfaz de usuario en un determinado estado, y el framework se encarga de actualizar la vista cuando el estado cambia.

|  |  |
| --- | --- |
| 1  2  3 | function saludo(usuario) {  return <h1>Hola, {usuario.nombre}!</h1>;  } |

En este ejemplo, se describe cómo debería ser el saludo en función de un objeto usuario, pero no se especifica cómo debe actualizarse el DOM (Document Object Model).

**Ventajas y desventajas:**

* **Ventajas:** El código declarativo tiende a ser más conciso y expresivo, lo que puede mejorar la legibilidad y reducir la cantidad de errores. También facilita la paralelización y optimización automática por parte del compilador o intérprete.
* **Desventajas:** La abstracción adicional puede llevar a una falta de control sobre los detalles finos de la ejecución, lo que podría ser problemático en situaciones donde el rendimiento es crítico.

Así mismo, la Programación Orientada a Objetos (POO) es un paradigma que organiza el software en "objetos", que son instancias de "clases". Una clase es una plantilla que define las propiedades (datos) y métodos (funciones) que son comunes a todos los objetos de ese tipo, los objetos interactúan entre sí mediante métodos, que permiten modificar el estado del objeto o realizar operaciones basadas en su estado actual.

**Conceptos clave de POO:**

* **Clase:** Una clase es una plantilla o modelo a partir de la cual se crean objetos. Define un conjunto de propiedades y métodos que serán comunes a todos los objetos de esa clase. Por ejemplo, una clase Coche podría tener propiedades como marca, modelo y color, y métodos como acelerar() y frenar().
* **Objeto:** Un objeto es una instancia de una clase, cada objeto tiene su propio conjunto de valores para las propiedades definidas en la clase, pero comparte la misma estructura y comportamiento. Por ejemplo, un objeto miCoche de la clase Coche podría tener la marca "Toyota", el modelo "Corolla" y el color "Rojo".
* **Propiedades:** Son los atributos o características de un objeto, definidos por su clase. En el ejemplo anterior, marca, modelo y color son propiedades de la clase Coche.
* **Métodos:** Son las funciones que se pueden ejecutar en un objeto. Los métodos definen el comportamiento del objeto. Por ejemplo, el método acelerar() podría aumentar la velocidad del coche.
* **Herencia:** La herencia es un mecanismo que permite a una clase derivar de otra, heredando sus propiedades y métodos, esto promueve la reutilización del código y la creación de estructuras jerárquicas. Por ejemplo, una clase Vehículo podría ser la clase base, y Coche y Motocicleta podrían heredar de Vehículo.

En lenguajes como Java, C++, y Python, la POO es fundamental. A continuación, un ejemplo básico en Python:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Coche:  def \_\_init\_\_(self, marca, modelo, color):  self.marca = marca  self.modelo = modelo  self.color = color  def acelerar(self):  print(f"El {self.marca} {self.modelo} está acelerando.")  mi\_coche = Coche("Toyota", "Corolla", "Rojo")  mi\_coche.acelerar() |

En este ejemplo, Coche es una clase con un constructor que inicializa las propiedades marca, modelo y color. El método acelerar() simula la acción de acelerar el coche.

Igualmente, la programación basada en eventos es un paradigma donde el flujo de la aplicación está determinado por eventos, como clics de usuario, pulsaciones de teclas, o mensajes de otros programas, los eventos son señales que indican que algo ha sucedido, y la aplicación responde ejecutando código en función de esos eventos.

**Características principales:**

* **Detección de eventos:** Los programas están diseñados para "escuchar" eventos y reaccionar cuando estos ocurren.
* **Asincronía:** A menudo, la programación basada en eventos se combina con la ejecución asincrónica, lo que permite que una aplicación responda a múltiples eventos sin bloquearse.
* **Handlers (manejadores):** Son funciones o métodos que se invocan en respuesta a un evento específico.

La programación basada en eventos es común en el desarrollo de interfaces gráficas de usuario (GUI), donde las acciones del usuario (como hacer clic en un botón) desencadenan eventos que la aplicación debe manejar. Por ejemplo, en JavaScript, los desarrolladores pueden usar eventos para responder a interacciones del usuario en una página web:

|  |  |
| --- | --- |
| 1  2  3 | document.getElementById("miBoton").addEventListener("click", function() {  alert("¡Botón clickeado!");  }); |

En este ejemplo, se añade un "listener" al botón con el ID miBoton, que escucha el evento click. Cuando el usuario hace clic en el botón, se ejecuta una función que muestra un mensaje de alerta.

Las aplicaciones móviles y las interfaces de usuario modernas dependen en gran medida de la programación basada en eventos, cada interacción del usuario, desde deslizar hasta tocar, se convierte en un evento que desencadena la lógica correspondiente en la aplicación, lo que permite una experiencia de usuario fluida e interactiva.