



PROGRAMACIÓN FUNCIONAL Y REACTIVA EN JAVA SOBRE LOS DESARROLLOS EN LATINOAMÉRICA

FUNCTIONAL & REACTIVE PROGRAMMING IN JAVA ON DEVELOPMENTS IN LATIN AMERICA

¹Fabiany Madrigal Salazar

²Omar Madrigal Salazar

RESUMEN

El constante cambio en los modelos de arquitectura de software ha generado la necesidad de soluciones ágiles, escalables y seguras que permitan a las empresas aumentar su rentabilidad. Para lograrlo, es esencial que todos los niveles del equipo, desde desarrolladores junior hasta arquitectos, conozcan los nuevos modelos, sus ventajas, desventajas y cómo implementarlos de forma que aporten valor al negocio. Los procesos de cambio de arquitectura suelen requerir la creación de un comité organizacional encargado de evaluar patrones, modelos y procesos adecuados a la estructura empresarial. Este enfoque facilita la incorporación de nuevas tecnologías y fomenta el aprendizaje continuo de los equipos de desarrollo. Entre los enfoques emergentes destacan la programación funcional y reactiva, que optimizan la ejecución de procesos por hilos, de manera síncrona o asíncrona, asegurando la continuidad de los flujos, la preservación de información y la consistencia en la definición de procesos. En Latinoamérica, la adopción de estos modelos evidencia un proceso de transformación tecnológica que impacta directamente en la eficiencia y competitividad de las organizaciones, mostrando cómo la apropiación y adaptación de nuevas arquitecturas son clave para enfrentar los retos del desarrollo de software contemporáneo.

PALABRAS CLAVE: arquitectura, concurrencia, desarrollador, funcional, hilos, java, micro servicios, reactiva, RxJava.

ABSTRACT

The constant evolution of software architecture models has created the need for agile, scalable, and secure solutions that enable companies to increase profitability. Achieving this requires that all team levels, from junior developers to architects, understand the new models, their advantages and disadvantages, and how to implement them in ways that add business value. Architecture change processes often necessitate the creation of an organizational committee responsible for evaluating patterns, models, and processes suitable for the company's structure. This approach facilitates the incorporation

¹Tecnología de Sistemas, Corporación Tecnológica Industrial Colombiana, IX semestre Ingeniería de Sistemas, Bogotá, Colombia, 1012349945@teinco.edu.co

²Tecnología de Sistemas, Corporación Tecnológica Industrial Colombiana, IX semestre Ingeniería de Sistemas, Bogotá, Colombia, 1012349945@teinco.edu.co

of new technologies and promotes continuous learning among development teams. Emerging approaches such as functional and reactive programming optimize process execution through threads, either synchronously or asynchronously, ensuring workflow continuity, information preservation, and consistency in process definition. In Latin America, the adoption of these models reflects a technological transformation that directly impacts organizational efficiency and competitiveness, demonstrating how the appropriation and adaptation of new architectures are essential to addressing the challenges of contemporary software development.

KEYWORDS: architecture, concurrency, developer, functional, threads, Java, microservices, reactive, RxJava.

INTRODUCCIÓN

Con los avances en el procesamiento y la incorporación del multicore en las CPU, los lenguajes de programación han debido adaptarse a la ejecución de procesos para aprovechar cada ciclo y transformar la mayor cantidad de datos por segundo. Según Warburton (2015), “en orden de habilitar la escritura de volúmenes de datos en librerías paralelas, Java necesitó un nuevo cambio en el lenguaje: expresiones lambda” (p. 1).

Desde 2001, con el surgimiento del multi-core, Castillo (2019) indica que “el primer procesador multinúcleo fue el IBM POWER4 con dos núcleos en un solo DIE y una frecuencia base de 1,1 GHz, fabricándose en el año 2001”. Este avance generó un elemento repetitivo en los desarrollos denominado concurrencia, ya que, anteriormente, las tareas de la CPU se ejecutaban de manera secuencial, provocando problemas de encolamiento y bajo rendimiento en las aplicaciones. Con los nuevos modelos de arquitectura, los lenguajes de programación comenzaron a mitigar estos cuellos de botella mediante la ejecución síncrona y asíncrona de tareas sobre hilos.

La concurrencia permite dividir las tareas y ejecutar procesos sin depender del orden. Castillo (2019) explica que “a este método de crear distintas tareas de forma simultánea de un programa se le llama hilos de procesos, hilos de trabajo, subprocesos o simplemente Threads en inglés”. Los procesos paralelos posibilitan ejecutar una misma tarea simultáneamente, lo que abre paso a soluciones ágiles, seguras y de alto rendimiento. Inicialmente, los hilos se utilizaron para controlar peticiones sincronizadas, asegurando flujos correctos en las aplicaciones y reduciendo los tiempos de respuesta. Además, dieron lugar a estructuras asíncronas donde el orden de ejecución no afectaba el resultado final.

Junto a la concurrencia surgieron nuevos patrones de programación, orientados a ejecutar tareas de manera organizada mediante disparadores. Algunos permiten ejecuciones parciales o completas de manera asíncrona sin comprometer el resultado final, lo que influyó en los nuevos modelos de pruebas de software. Sin embargo, dentro de estos flujos de prueba se identificó un desafío: la sobrecarga de la CPU. Según CodeGuru (2007), “cada hilo en software requiere de memoria virtual para las estructuras privadas y almacenadas de estructura de datos”, lo que generaba un alto consumo de recursos y retos en los patrones de diseño y arquitecturas emergentes. Una solución consistió en el manejo de interfaces y programación laxa mediante funciones.

En 2005, con la disponibilidad comercial de procesadores dual-core para escritorio (Castillo, 2019), los desarrolladores accedieron a tecnologías multi-core con menos restricciones, iniciando una nueva era de autonomía para decidir buenas prácticas, patrones de arquitectura y modelos de programación. En 2006, Java 6 incorporó soporte para hilos, anotaciones y la interfaz Runnable, según Jadeite (2008): “La otra manera de crear un hilo es declarar una clase que implementa la interface Runnable. La clase implementa el método run. Una instancia de la clase puede ser localizada, pasando un argumento al crear el hilo, y se inicia” (p. X). Además, se implementaron pools de conexiones y schedulers para la programación concurrente y paralela de tareas.

Con la evolución de JDK y frameworks como Spring, Norén (2019) indica que “en 2014 aparece el release de Spring Boot 1.0, que habilita el desarrollo más rápido en aplicaciones Spring al integrar la configuración automática en los proyectos” (p. X). Esta innovación permitió la gestión eficiente de peticiones síncronas y asíncronas, así como la inyección de dependencias mediante gestores como Maven o Gradle, optimizando el consumo de recursos y mejorando el rendimiento de las aplicaciones.

A partir de esta evolución, surge la pregunta: ¿cuáles son los mejores modelos orientados a hilos? Una respuesta es trabajar con un modelo funcional, donde cada función sea accesible desde cualquier paquete del programa, facilitando la creación de componentes con una sola responsabilidad y cumpliendo principios SOLID (Singleton, Open-Close, Liskov, Interface Segregation y Dependency Injection). Otra opción es implementar un modelo de programación reactivo, orientado a eventos asíncronos y promesas de objetos, que permite un procesamiento secuencial y asíncrono, aumentando la agilidad y la estructuración de los procesos.

En Latinoamérica, la adopción de modelos funcionales y reactivos en Java ha crecido durante los últimos tres años, favoreciendo el desarrollo de aplicaciones livianas, rápidas y reutilizables, alineadas con metodologías ágiles. La creación de MVPs (Minimum Viable Product) ha generado un crecimiento exponencial en la demanda de desarrolladores, aunque algunos consideran que podría existir una burbuja de oferta. No obstante, los constantes cambios tecnológicos y las necesidades de los usuarios impulsan la evolución de la programación, generando un aprendizaje continuo y planteando a los desarrolladores la pregunta: ¿cuál es la siguiente tecnología que deberían aprender?

METODOLOGÍA

Materiales y métodos

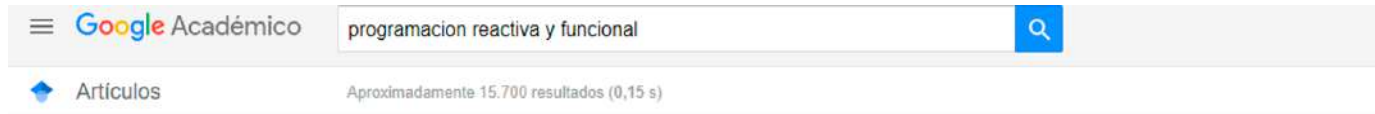
El artículo de revisión se desarrolló bajo una metodología de investigación descriptiva, cuyo objetivo es indagar en las principales características de los modelos de programación funcional y reactivo aplicados al lenguaje Java. Para ello, se realizó un análisis exhaustivo que abarcó desde la revisión de los Javadocs de cada versión de JDK y SDK, hasta la consulta de artículos académicos y libros especializados, con el fin de identificar los elementos que motivan a empresas y desarrolladores a adoptar estos nuevos modelos de desarrollo. Asimismo, se busca determinar las posibles direcciones que pueden seguir los desarrollos empresariales en función de dichas elecciones tecnológicas.

Los elementos que se abordarán en este artículo se centran en el aprendizaje sobre los nuevos modelos de programación, la adaptación de dichos modelos a desarrollos legados o deprecados, la implementación de modelos ajustados a la infraestructura definida por el cliente, la construcción de software sobre nuevas propuestas de arquitectura y el tiempo requerido para la adopción efectiva de un nuevo modelo de programación.

Análisis bibliográfico

Para la elaboración del presente artículo, se recopilaron fuentes provenientes de libros, artículos académicos y sitios web especializados. En total, se identificaron aproximadamente 17,500 fuentes con información relevante sobre el tema seleccionado, “Programación reactiva y funcional”, correspondientes a diversos lenguajes de programación y autores que han profundizado en el tema. De esta cantidad, se seleccionaron 50 fuentes que se consideraron pertinentes para la construcción y desarrollo del presente artículo, garantizando así la validez y relevancia de la información presentada (véase figura 3).

Figura 3. Criterios de búsqueda



Nota. Elaboración propia.

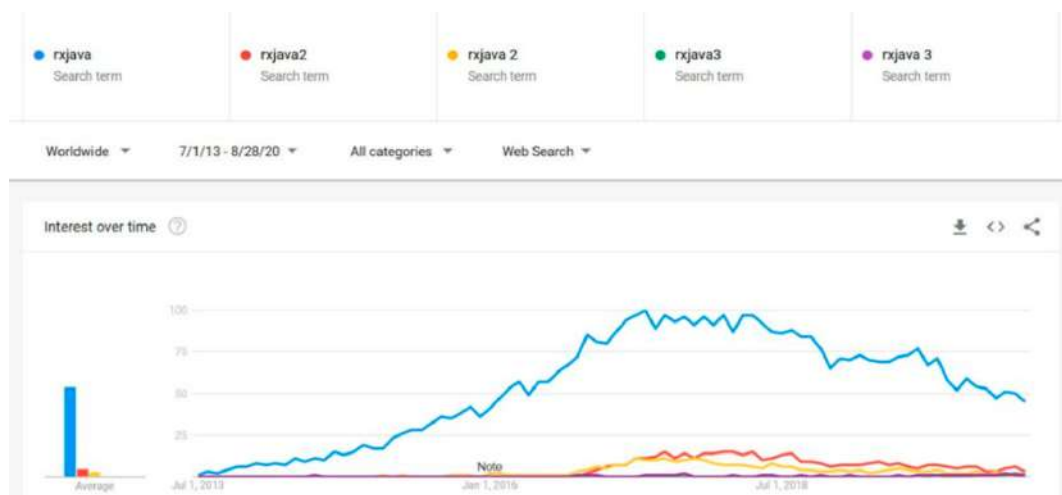
Dentro de las fuentes comunes de información e instituciones que implementan modelos de programación reactiva y funcional, se identificó que aproximadamente 20 instituciones poseen una cantidad significativa de referencias, libros y artículos relacionados con el tema. Sin embargo, al analizar estas fuentes, se observó que solo un número reducido resultó útil para la construcción del presente artículo, dado que el enfoque de investigación y análisis documental se centró en Latinoamérica.

Del listado de fuentes seleccionadas se incluyeron artículos académicos, monografías, publicaciones web e investigaciones que permiten recopilar información sobre los procesos de desarrollo y la evolución de la programación funcional, así como el uso de librerías especializadas en reactividad y orientación a eventos.

En la Figura 4 se observa la tendencia de búsqueda por palabras clave relacionadas con la librería RxJava, desde su primera versión hasta la más reciente, mostrando un auge en su implementación y estudio. Esta tendencia se complementa con la información presentada en la Figura 5, donde se analiza la búsqueda en el principal sitio de preguntas y respuestas de la comunidad global de desarrollo, evidenciando un incremento de usuarios que emplean las librerías para distintos propósitos, ya sea en aplicaciones móviles, aplicaciones web o sitios desarrollados bajo el patrón MVC (Model, View, Controller).

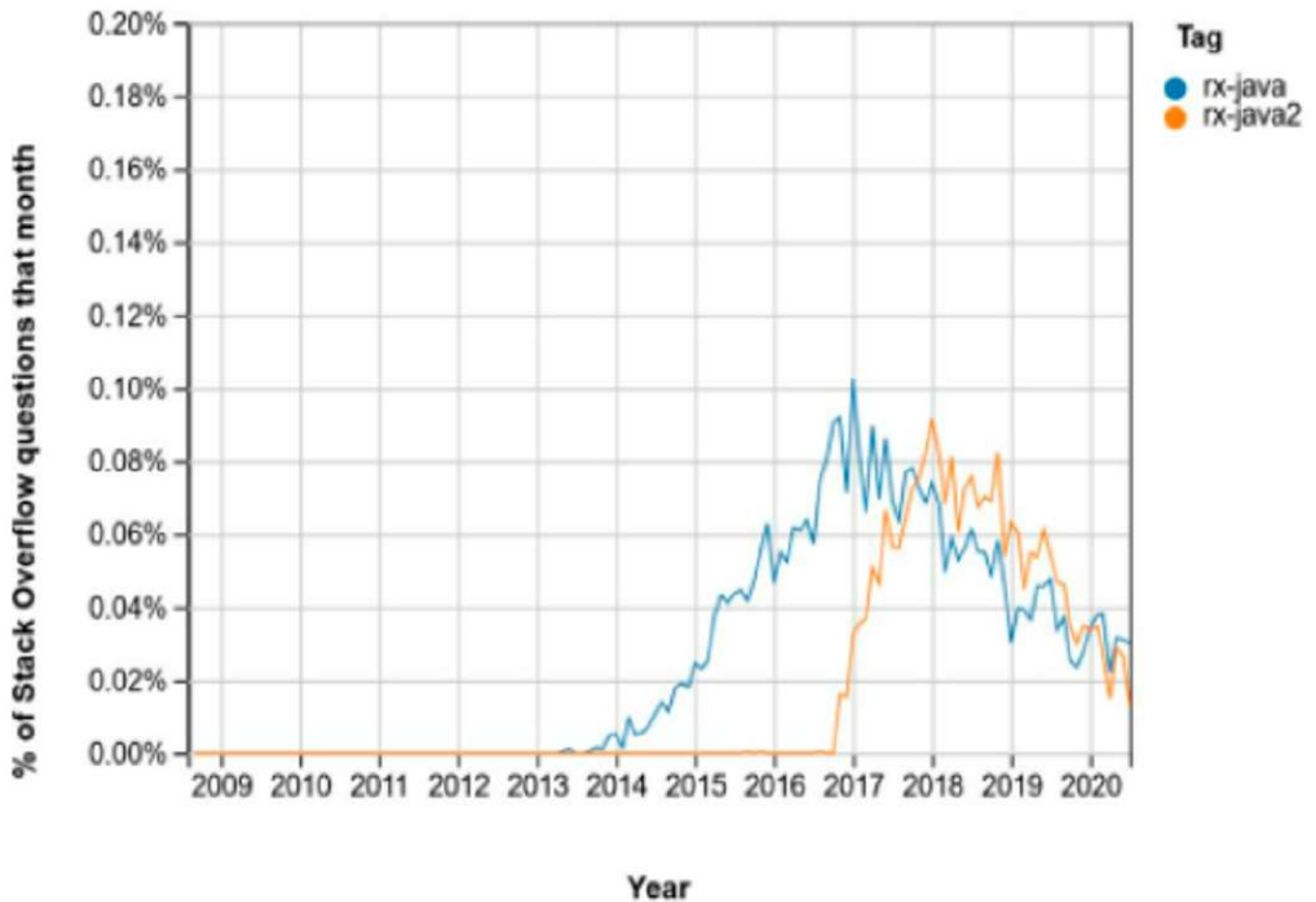
Vasilily (2020) indica que “lo más importante, presentando RxJava y tareas asíncronas como solo alternativas a una falsa dicotomía. Como escribí en mi artículo sobre concurrencia y marcos de trabajo, las clases sobre hilos y las piscinas de hilos siempre han sido una muy buena opción” (p. 18). Esto evidencia que existen alternativas para la implementación de procesos asíncronos mediante hilos, lo que permite un uso eficiente del procesador en diversos tipos de dispositivos o servidores, optimizando así la programación funcional y reactiva.

Figura 4. Resultados por keyword RxJava



Nota. Adaptado de Vasilily (2020).

Figura 5. Resultados de preguntas realizadas en StackOverFlow



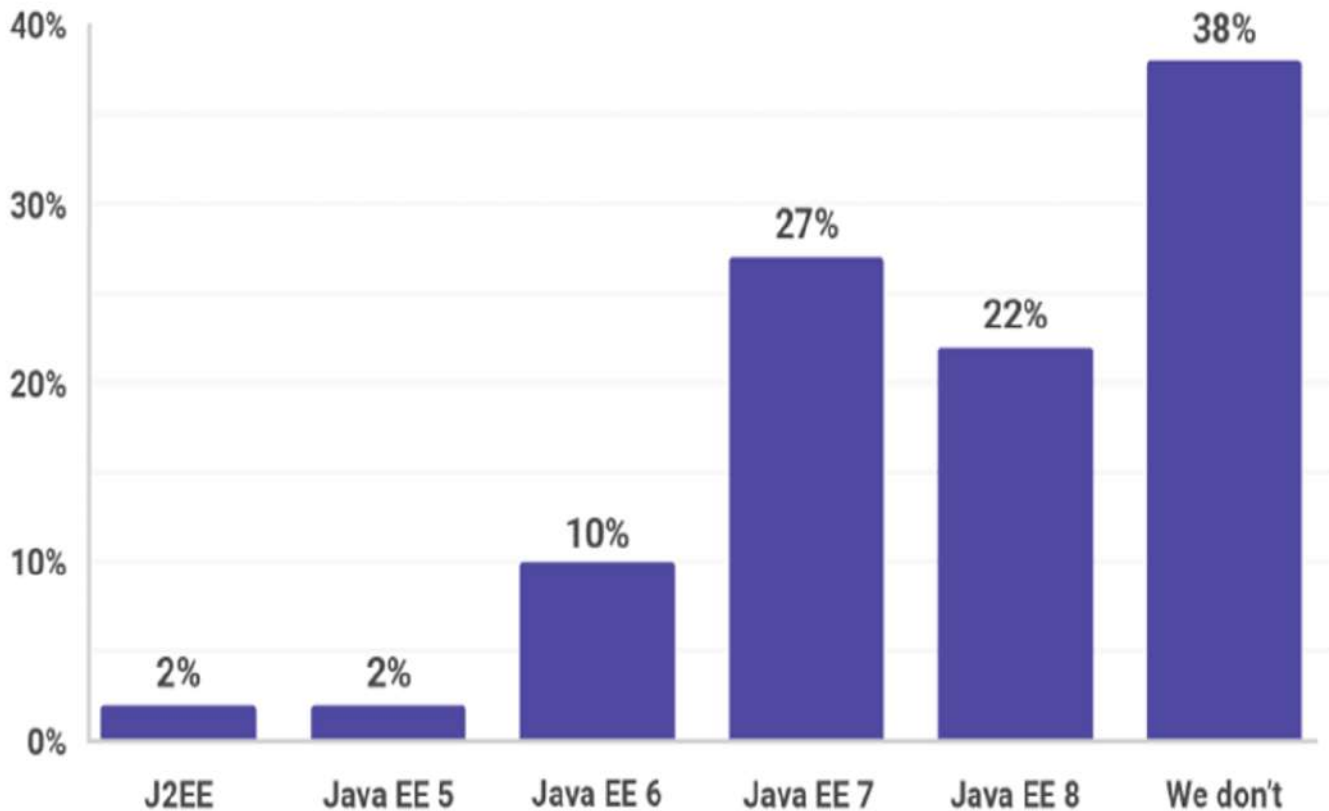
Nota. Adaptado de Vasilliy (2020).

En las fuentes analizadas sobre los procesos basados en Java en entornos productivos de Latinoamérica, se observa que los ecosistemas a partir de la versión 8 comienzan a implementar de manera significativa modelos de programación reactiva y funcional. Los porcentajes de adopción por versión son los siguientes:

- 2 % corresponde a la versión Java J2EE.
- 5 % corresponde a la versión Java JEE 5.
- 10 % corresponde a la versión Java JEE 6.
- 27 % corresponde a la versión Java JEE 7.
- 22 % corresponde a la versión Java JEE 8.
- 38 % corresponde a versiones no identificadas, que se asume son superiores a la 8.

Estos datos evidencian que la adopción de versiones recientes de Java, especialmente a partir de la versión 8, ha facilitado la incorporación de paradigmas funcionales y reactivos en los desarrollos empresariales en la región, reflejando una tendencia creciente hacia la modernización de los sistemas y la optimización de procesos concurrentes y asíncronos (véase figura 6).

Figura 6. El estado del ecosistema Java en 2018



Nota. Adaptado de campusMVP (2018).

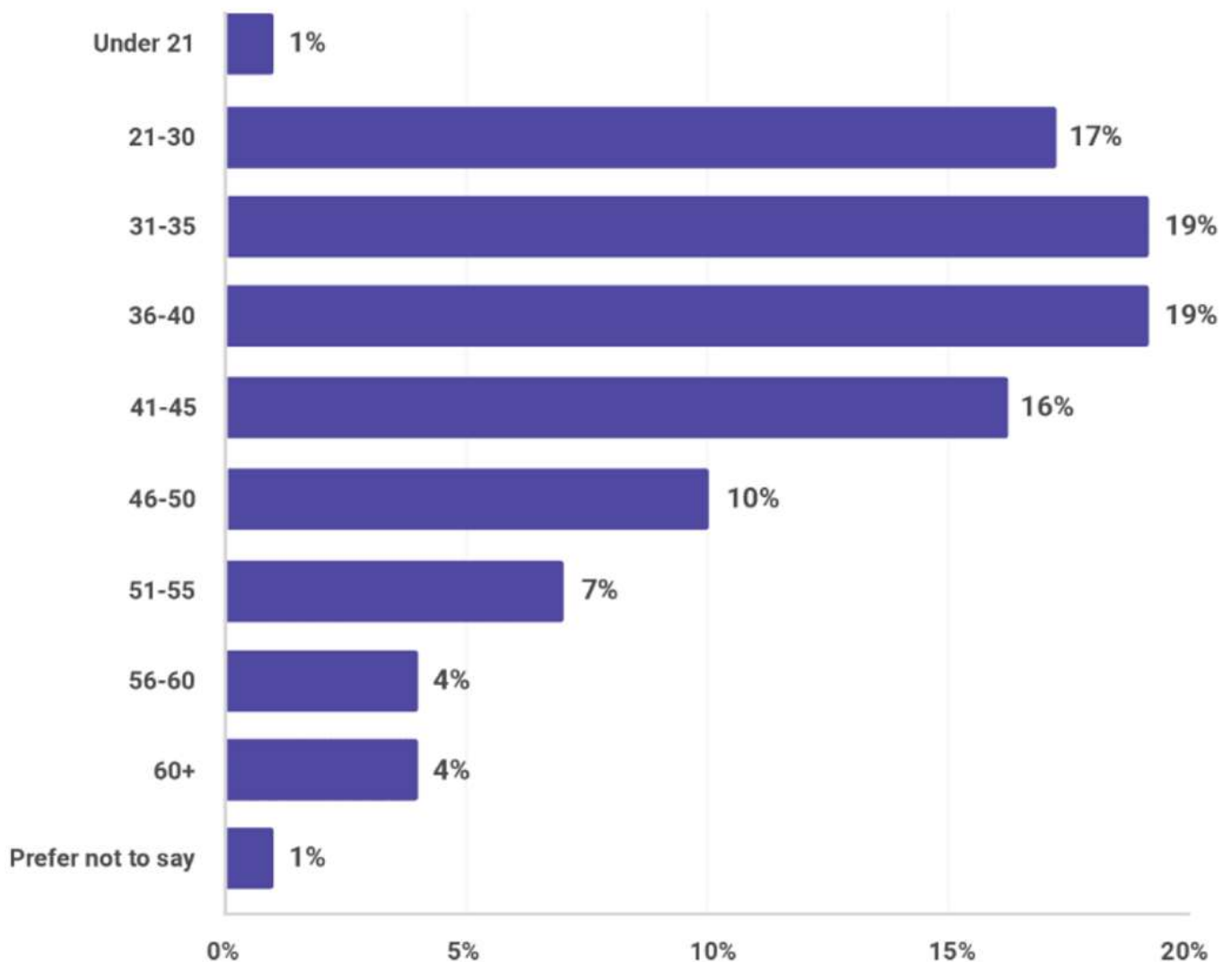
Autoaprendizaje de nuevos modelos de programación

Los nuevos modelos de programación representan un desafío para aquellos desarrolladores que inician su trayectoria profesional con conocimientos de versiones anteriores de un lenguaje y que, por diversas razones, han decidido mantenerse en dichas versiones durante años. Sin embargo, tarde o temprano se ven en la necesidad de aprender nuevos modelos de programación o incluso nuevos lenguajes. En este contexto, se podría pensar que los desarrolladores de mayor edad tienen dificultades para adoptar nuevas tecnologías, pero según CampusMVP (2018), “Estos son los rangos de edad de los programadores de Java encuestados: la mayoría está entre 30 y 45 años, pero los mayores de 50 constituyen un número considerable (25 %), y en general hay muchos años de experiencia en la muestra” (párr. 3). Esto indica que el rango de edad de quienes aplican y aprenden nuevos modelos de programación es amplio, y que el autoaprendizaje se convierte en una herramienta fundamental para el desarrollo profesional, a través de plataformas gratuitas o certificadas, tanto en modalidad virtual como presencial.

Uno de los elementos más relevantes del aprendizaje autónomo es la gamificación, que permite adquirir conceptos, lógicas y metodologías de manera interactiva, facilitando la retención de la información de forma más efectiva. Plataformas como CodeWars, CodeGame y CodeFight aplican este modelo de competencia para promover la práctica y comprensión de la programación. Según Vives (2021), “El autoaprendizaje gamificado para la comprensión de la programación se basa en los factores motivacionales, cognitivos y la didáctica gamificada, como elementos que favorecen las condiciones para el autoaprendizaje de la programación” (p. 48, párr. 3).

Adicionalmente, los cursos de formación presentan una relación directa de 1 a 3 horas de práctica por cada concepto aprendido, y la inversión de tiempo necesaria para un aprendizaje adecuado oscila entre 24 y 200 horas, dependiendo de la motivación del desarrollador, su disponibilidad de tiempo y los recursos físicos o tecnológicos con los que cuente. Según la estadística de desarrolladores por edad (Figura 7), la mayoría se encuentra entre los 21 y 40 años, representando un 55% del total, lo que indica que gran parte de los participantes son jóvenes adultos con alta adaptabilidad a nuevas tecnologías, mientras que solo un pequeño porcentaje supera los 50 años, mostrando que la experiencia previa puede jugar un papel importante en la optimización del aprendizaje.

Figura 7. Estadística desarrolladores por edad



Nota. Adaptado de campusMVP (2018).

Actualmente, existen numerosas plataformas que facilitan el autoaprendizaje en programación, ofreciendo cursos impartidos tanto por desarrolladores senior como por instructores especializados que estructuran una malla curricular completa para el desarrollo de software. Esta tendencia surge, en parte, debido a experiencias insatisfactorias en instituciones educativas

tradicionales, donde algunos instructores presentan conocimientos limitados sobre los temas que imparten, lo que afecta la calidad de la educación presencial. En este contexto, los desarrolladores suelen recurrir a plataformas como Udemy, Udacity, Coursera, Webinar, Plazzi, entre otras, donde el aprendizaje es más enfocado y evaluado de acuerdo con las necesidades del profesional, con costos accesibles (Vita, 2021). Según el autor, “Las diferentes instituciones educativas buscan que el aprendizaje de los alumnos en una plataforma educativa los lleve a aprender de manera autónoma y lograr con ello la metacognición y un aprendizaje significativo” (p. 2-3).

La iniciativa de aprendizaje autónomo es frecuente entre los desarrolladores, quienes suelen ser personas curiosas y con interés por explorar nuevas tecnologías, mejorar su desempeño y salir de su zona de confort después de varios años en el mismo entorno. En palabras de algunos expertos, “La experiencia no es hacer la misma tarea por un largo tiempo”, lo que refleja el deseo constante de actualización. Además, los desarrolladores se adaptan a nuevas tecnologías y tendencias, migrando cuando identifican mejores perspectivas profesionales (Herranz, s.f., p. 8). Este proceso de autoaprendizaje permite que los conceptos y paradigmas de programación se adopten de manera rápida y efectiva.

El aprendizaje en línea se estructura generalmente mediante guías y ejercicios que permiten al usuario avanzar gradualmente, con puntos de control similares a exámenes asistidos, aumentando la atención y fomentando un entorno interactivo y colaborativo. Gonzales, Lopez, Trujillo y Rojas (2017) señalan que estas metodologías favorecen “el aumento de la atención en los estudiantes y la creación de un entorno interactivo y colaborativo” (p. 14). Este enfoque permite repasar conceptos sin depender de recursos externos y reforzar la comprensión de manera efectiva.

Los entornos virtuales se han convertido en un recurso recurrente en la educación digital, aunque la elección de la plataforma depende de múltiples factores, desde políticas institucionales hasta requisitos mínimos para el curso (Otero & Alma, 2017, p. 98). Entre los criterios para seleccionar una plataforma se encuentran el precio, los beneficios internos y la posibilidad de obtener certificaciones reconocidas por entidades regulatorias.

En cuanto a la calidad del autoaprendizaje, Martelo, Franco y Oyala (2020) indican que “medir la calidad de educación virtual implica grandes retos, debido a la alta cantidad de variables que se deben considerar para su estudio” (p. 359). Esto evidencia que, aunque los cursos en línea permiten el acceso a conocimientos especializados, no todos los instructores cuentan con formación pedagógica suficiente para garantizar un flujo adecuado en la enseñanza.

Finalmente, además de los cursos, existen plataformas que ofrecen retos de programación para consolidar habilidades prácticas y medir estándares de calidad mediante herramientas como SonarQube. Según Zinovieva et al. (2021), “El término ‘plataforma de código online’ es usado para denotar el concepto de los simuladores online para dominar y consolidar las habilidades de programación” (p. 6). Plataformas como CodeWars, CodeFight y SoloLearn permiten a los desarrolladores practicar y fortalecer competencias adquiridas durante el proceso de autoaprendizaje.

Tiempo promedio en que un desarrollador aprende y adopta un nuevo modelo de programación dependiendo su seniority

Al comprender el alcance de los procesos de autoaprendizaje, se observa que contar con una base sólida permite avanzar de manera más rápida y eficaz. Esto se debe a que mantener estructuras fundamentales facilita el dominio progresivo de la sintaxis y de las particularidades de cada lenguaje de programación. En palabras de Casey (2021), “muchos lenguajes de programación pueden parecer similares, no hay dos lenguajes que se comporten de la misma manera. Los desarrolladores y arquitectos deben observar de cerca las fortalezas y debilidades de cada opción” (p. 2). Esta afirmación destaca que, aunque las bases sean parecidas, tanto la sintaxis como la implementación pueden variar significativamente, incluso al ejecutar sentencias sencillas, como se ilustra en la Figura 8.

Figura 8. Hola mundo en diversos lenguajes



Nota. Tomado de Pinterest (s.f.).

Entrando en el tema, el tiempo de aprendizaje tiende a ser relativo, ya que depende de diversos factores, como la disponibilidad de tiempo, la complejidad del lenguaje y la urgencia de aprendizaje. Al considerar un curso completo, que abarca desde los conceptos básicos hasta los avanzados de un lenguaje, framework, modelo o paradigma de programación, se estima que el tiempo requerido oscila entre 24 y 60 horas. No obstante, en muchas ocasiones algunas lecciones se omiten debido a conocimientos previos que el estudiante posee sobre estructuras o conceptos similares en otros lenguajes. Según Caroline (2020), “pueden proceder a lo largo de un curso a su propio paso y en su propio tiempo. No fuerza a tus empleados a cumplir con un horario difícil. Tienen el poder de ajustar su esquema de estudio a su situación” (p. 5). Esto permite flexibilidad, pero también implica que la disponibilidad de tiempo puede convertirse en un factor limitante.

Al considerar los tiempos de aprendizaje, es posible establecer una tabla de referencia relativa para dominar un elemento de manera efectiva, tomando como ejemplo el aprendizaje de un segundo idioma (ver Figura 9). De manera análoga, los lenguajes de programación pueden clasificarse según el nivel de seniority del desarrollador. La escala A corresponde al nivel Junior, con experiencia de 0 a 2 años; la escala B al nivel Semi Senior, con experiencia de 2 a 5 años; y la escala C al nivel Senior, con experiencia superior a 5 años.

Figura 9. Horas por nivel de inglés

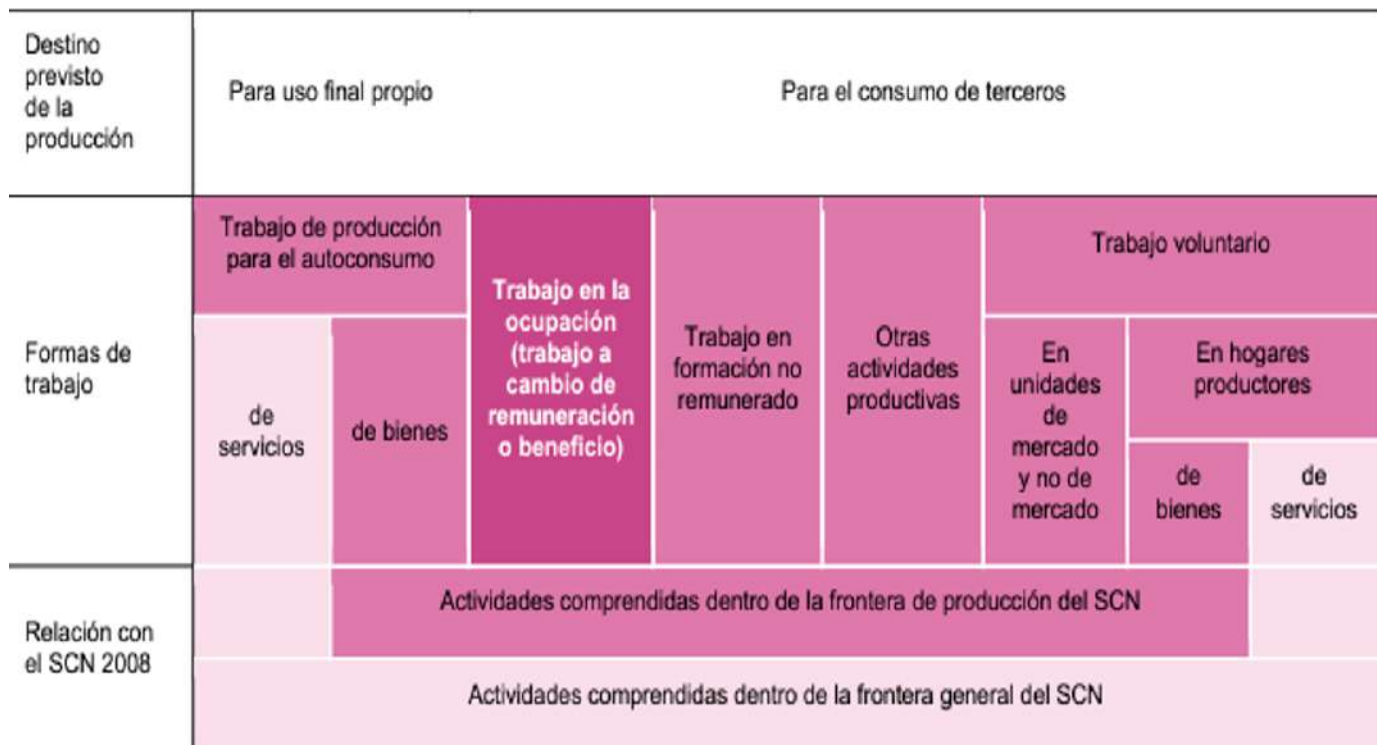
CEFR Level	Cambridge English Exam	Number of Hours (approximate)
C2	C2 Proficiency - previously known as Cambridge English: Proficiency (CPE)	1,000 - 1,200
C1	C1 Advanced - previously known as Cambridge English: Advanced (CAE)	700 - 800
B2	B2 First - previously known as Cambridge English: First (FCE)	500 - 600
B1	B1 Preliminary - previously known as Cambridge English: Preliminary (PET)	350 - 400
A2	A2 Key - previously known as Cambridge English: Key (KET)	180 - 200
A2	A1 Starters/Movers - previously known as Cambridge English Starters/Movers (YLE)	90 - 100

Nota. Tomado de Diagonal English (s.f.).

Al tener la noción del dominio de un lenguaje, se puede estimar que el tiempo real de aprendizaje excede las horas del curso, utilizando un cálculo proporcional. Por ejemplo, aplicando una regla de tres y multiplicando las horas del curso por un factor de 3,5, se obtiene el tiempo efectivo invertido en consolidar los conocimientos básicos. De esta manera, un curso de 60 horas implicaría un total de 210 horas de dedicación real para adquirir un dominio adecuado del contenido (Naciones Unidas, 2016).

A partir de este cálculo, es posible identificar los períodos de tiempo libre de los cuales un individuo podría disponer para realizar el curso, considerando la organización de la jornada según las actividades humanas, que se clasifican en tres categorías: trabajo remunerado, trabajo no remunerado (componentes del trabajo total) y actividades no productivas o personales (Naciones Unidas, 2016, p. 13). La Figura 10 ilustra la distribución de los tiempos disponibles, mostrando cómo se puede organizar el tiempo de manera eficiente para el aprendizaje.

Figura 10. Diagrama de formas del trabajo

Formas de trabajo en relación con el Sistema de Cuentas Nacionales (SCN), 2008

Nota. Elaboración propia. Adaptado de Naciones Unidas (2016).

Como conclusión preliminar del tema, se puede afirmar que, dependiendo de la saturación de actividades, las personas pueden destinar un tiempo determinado para el aprendizaje. Algunas empresas ofrecen dentro de la jornada laboral de una a dos horas para el autoaprendizaje y el refuerzo de materias que se manejan a diario pero que no se dominan completamente. Considerando un cálculo aproximado de 210 horas disponibles para completar un curso de 60 horas y adicionando dos horas en el tiempo personal, se obtiene un total de cuatro horas diarias en el mejor de los casos. Bajo esta estimación, se necesitarían aproximadamente 54 días para completar el aprendizaje mediante un proceso autodidacta (Prince, 2020, p. 4). Este proceso puede prolongarse durante meses, especialmente para cursos o actividades que requieren un esfuerzo considerable.

Si se considera a un desarrollador con nivel junior en un lenguaje de programación, la adaptación a un nuevo lenguaje puede requerir entre una y siete horas diarias de aprendizaje, debido a la cantidad de términos y conceptos nuevos que deben integrarse. Además, un desarrollador junior aún no posee la capacidad de relacionar conceptos previos con los nuevos, lo que ralentiza el aprendizaje y genera sobreesfuerzo (Méndez, 2020, p. 9). En este nivel, el conocimiento real es limitado y el desarrollador solo domina conceptos básicos, lo que puede crear un bloqueo al enfrentar un nuevo lenguaje de programación simultáneamente.

Por otro lado, los desarrolladores con mayor seniority pueden identificar los puntos clave del aprendizaje continuo a lo largo de su carrera. Sin embargo, uno de los problemas principales es la disponibilidad de tiempo. Las tareas diarias y responsabilidades profesionales reducen las horas libres, por lo que muchos recurren a horarios nocturnos para el

autoaprendizaje, durmiendo entre cinco y seis horas por costumbre o necesidad, aunque el rango recomendado es de seis a ocho horas (Patrick, 2022, p. 4).

Es importante considerar que la falta de tiempo y espacio para el aprendizaje puede afectar los ciclos de sueño, generando impactos negativos en la salud. Asimismo, factores como desórdenes del sueño o hiperactividad pueden alterar el flujo de energía, desviándolo hacia otras actividades. Permanecer largas horas frente al computador sin descanso adecuado puede provocar fatiga, sobrealimentación con productos poco saludables y reducción de la calidad del sueño, lo que afecta tanto el aprendizaje como la capacidad de desarrollar código de manera eficiente (Lores, 2018, p. 11). Por ello, se recomienda planificar el autoaprendizaje de forma equilibrada, asegurando períodos de descanso suficientes para mantener la salud física y mental, lo cual repercute directamente en la calidad del aprendizaje y del desarrollo profesional.

Costo y beneficio de migración de aplicaciones a nuevos modelos de programación tecnológicas

Para las empresas, uno de los mayores desafíos es mantener los procesos en un modelo óptimo tanto en tiempo como en seguridad. Si se presentan fallas en cargas, tiempos de respuesta o seguridad, se generan pérdidas significativas para la organización, afectando su reputación o incluso provocando pérdida masiva de clientes. Un ejemplo es el proceso bancario, donde el core (COR) normalmente se encuentra en lenguajes como COBOL o AS400, los cuales siguen siendo altamente seguros y rápidos para procesos de backend. Sin embargo, la parte visible para el cliente requiere un flujo más flexible, empleando diversos lenguajes que se adapten a la funcionalidad requerida y minimicen problemas durante su ejecución (López & Maya, 2017, p. 2).

Las migraciones empresariales son procesos complejos y delicados. Desde las áreas de arquitectura de software, se busca minimizar riesgos y vulnerabilidades que se acumulan con el tiempo, ya que arquitecturas que eran eficientes hace diez años pueden no ajustarse a los estándares actuales de seguridad o a los procesos de negocio (Torres, Camacho, Solano, & León, 2020, p. 4). Migrar a sistemas más óptimos permite integrar nuevas plataformas, mejorar el rendimiento de las aplicaciones y facilitar el mantenimiento, haciendo que los procesos sean escalables y ágiles.

Durante la adaptación de nuevos modelos de arquitectura, es fundamental conocer cómo se pueden ajustar a la infraestructura existente, supliendo necesidades y definiendo procesos de escalabilidad dentro de la organización. La creación de componentes sobre arquitecturas nuevas permite eliminar elementos obsoletos y construir soluciones más eficaces, optimizando la lógica de negocio (Velázquez, Flores, Ibarra, Astorga, Aguilar, Ramírez, & López, 2021, p. 3).

Dependiendo de la infraestructura, los procesos de migración pueden variar considerablemente. Por ejemplo, cambiar un core bancario basado en COBOL o AS400 puede llevar años, mientras que migrar una aplicación monolítica a microservicios para un MVP puede completarse en meses (EvaluandoSoftware, 2021, p. 3). La arquitectura de software debe adaptarse a los objetivos de negocio, garantizando integración eficiente y rendimiento óptimo de todos los componentes.

En la actualidad, las áreas de DevOps juegan un papel crucial en la integración continua, gestionando despliegues en distintos ambientes y asegurando la correcta configuración y funcionamiento de la arquitectura de software. Esto requiere la colaboración de todo el personal involucrado, desde el desarrollo hasta la finalización del proyecto (Mamani, Pino, & Gonzalez, 2020, p. 11).

Un factor crítico durante los cambios tecnológicos es la resistencia al cambio. Al modificar procesos, herramientas o la cultura organizacional, los integrantes pueden percibir amenazas a su zona de confort. Por ello, es necesario

implementar estrategias de inclusión, comunicación y adaptación que minimicen los bloqueos y faciliten la transición (Hernández, 2021, p. 1).

El cambio cultural en las organizaciones permite no solo mejorar los procesos de negocio, sino también potenciar el desarrollo de los integrantes. Una cultura ágil fomenta la creación de valor en tiempos cortos, con soluciones a la medida y respetando los recursos de la organización (Rincón Rodríguez & Aldana Bautista, 2021, p. 3). La falta de compromiso por parte del personal puede dificultar la implementación de nuevos procesos y generar bloqueos en los flujos de negocio.

Finalmente, dentro de las metodologías ágiles, Scrum se ha consolidado como una herramienta efectiva para la gestión de proyectos. Permite realizar un seguimiento por integrante, por equipo y por releases, facilitando la refactorización del flujo de negocio y la construcción de aplicaciones de manera incremental. Esta metodología ofrece resultados concretos y medibles, en contraste con enfoques tradicionales como Waterfall, promoviendo eficiencia y efectividad en proyectos de desarrollo de software (Scrum Project Implementation, p. 4, 66).

CONCLUSIONES

Los nuevos modelos de programación representan herramientas efectivas para el desarrollo de soluciones óptimas y concisas, aportando un valor significativo a las organizaciones que los implementan. Sin embargo, al ser tecnologías recientes, resulta complejo y desafiante encontrar desarrolladores con conocimientos suficientes para aplicarlas de manera eficiente.

Como alternativa, se propone la implementación de capacitaciones dentro del horario laboral dirigidas a los desarrolladores de software. Esta estrategia no solo busca dotar a los profesionales de competencias actualizadas, sino también generar valor agregado para la organización, favoreciendo la retención de talento y reduciendo la rotación de personal.

La programación funcional y reactiva permite la construcción de procesos ágiles, controlados mediante hilos, y la gestión eficiente de cronogramas internos de ejecución (Schedule) a nivel de arquitectura. Esto optimiza el uso de recursos, evitando sobrecargas y minimizando costos en los procesos organizacionales.



REFERENCIAS



- Al Mamun, A., Khanam, A., Grahan, H., & Feldt, R. (2010). *Comparing four static analysis tools for Java concurrency bugs* [PDF]. http://www.robertfeldt.net/publications/grahn_2010_comparing_static_analysis_tools_for_concurrency_bugs.pdf
- Caroline. (2021). *La lista completa de las ventajas y desventajas de los cursos online*. Easy LMS. <https://www.easy-lms.com/es/centro-de-conocimiento/cursos/ventajas-y-desventajas-de-los-cursos-online/item10402>
- CampusMVP. (2018). *El estado del ecosistema Java en 2018*. <https://www.campusmvp.es/recursos/post/el-estado-del-ecosistema-java-en-2018.aspx>
- Castillo, J. (2019). *Procesador multinúcleo: Qué es y para qué sirve*. Profesional Review. <https://www.profesionalreview.com/2019/07/14/procesador-multinucleo/>
- Casey, K. (2021). *Comparación de 6 de los principales lenguajes de programación*. <https://www.computerweekly.com/es/consejo/Comparacion-de-6-de-los-principales-lenguajes-de-programacion>
- CodeGuru Staff. (2007). *Why too many threads hurts performance and what to do about it*. <https://www.codeguru.com/cplusplus/why-too-many-threads-hurts-performance-and-what-to-do-about-it/>
- Diagonal English. (s.f.). *¿En cuánto tiempo se aprende inglés?* [Imagen]. <https://diagonalenglish.com/en-cuanto-tiempo-se-aprende-ingles/>
- EvaluandoSoftware. (2021). *La arquitectura empresarial prepara a la compañía para los cambios del mercado*. <https://www.evaluandosoftware.com/la-arquitectura-empresarial-prepara-la-compania-los-cambios-del-mercado/>
- Garcia, M. (2022). *Implementación de una arquitectura micro-frontend para optimizar el desarrollo y despliegue de la aplicación web del sistema de información universitaria de la SUNEDU*. http://cybertesis.unmsm.edu.pe/bitstream/handle/20.500.12672/17887/Garcia_chm.pdf?sequence=1&isAllowed=y
- Gonzales, E., Lopez, A., Trujillo, V., & Rojas, R. (2017). *Estrategia didáctica de enseñanza y aprendizaje para programadores de software*. <https://www.ride.org.mx/index.php/RIDE/article/view/402/2079#info>
- Hernández, S. (2021). *Estrategias para afrontar la resistencia al cambio en las organizaciones*. <https://repository.unimilitar.edu.co/bitstream/handle/10654/39044/HernándezGamboaSandraLiliana2021.pdf?sequence=2&isAllowed=y>



REFERENCIAS



- Herranz, X. (2020). *20 años después quiero volver a programar: Qué grandes cambios ha habido y por dónde empezar*. <https://www.xataka.com/otros/20-anos-despues-quiero-volver-a-programar-que-grandes-cambios-ha-habido-donde-empezar>
- Lores, M. (2018). *El síndrome de "Burnout" o el desgaste profesional del programador*. <https://www.campusmvp.es/recursos/post/el-sindrome-de-burnout-o-el-desgaste-profesional-del-programador.aspx>
- López, D., & Maya, E. (2017). *Arquitectura de software basada en microservicios para desarrollo de aplicaciones web*. <http://138.59.12.47/bitstream/10786/1277/1/93%20Arquitectura%20de%20Software%20basada%20en%20Microservicios%20para%20Desarrollo%20de%20Aplicaciones%20Web.pdf>
- Mamani, Z., Pino, L., & Gonzalez, C. (2020). *Arquitectura basada en microservicios y DevOps para una ingeniería de software continua*. <https://www.redalyc.org/journal/816/81665362014/html/>
- Martelo, R., Franco, D., & Oyala, P. (2020). *Factores que influyen en la calidad de la educación virtual*. <http://www.revistaespacios.com/a20v41n46/a20v41n46p29.pdf>
- Méndez, P. (2020). *Senior vs Junior! (Niveles de programadores)*. <https://ed.team/blog/senior-vs-junior-niveles-de-programadores>
- Naciones Unidas. (2016a). *Clasificación de actividades de uso del tiempo para América Latina y el Caribe*. https://repositorio.cepal.org/bitstream/handle/11362/39624/4/S1600508_es.pdf
- Naciones Unidas. (2016b). *Diagrama de formas del trabajo* [Imagen]. <https://www.un.org/es/sections/issues-depth/work/index.html>
- Norén, A. (2019). *Un poco de historia sobre Spring*. <https://javadesde0.com/un-poco-de-historia-sobre-spring/>
- Nurkiewicz, T., & Christensen, B. (2017). *Reactive programming with RxJava: Creating asynchronous, event-based*. https://books.google.es/books?hl=es&lr=&id=y4Y1DQAAQBAJ&oi=fnd&pg=PR4&dq=rxjava+programming+paradigm&ots=Di9YY-X5Rh&sig=fFE3UwM1dbi2irII45n_I0aPQIo#v=onepage&q&f=false
- Otero, E., & Alma, D. (2017). *Plataformas virtuales de aprendizaje en la educación superior*. <https://is.uv.mx/index.php/IS/article/view/2545/4454>



REFERENCIAS



- Patrick. (2022). *Sleep more, code more*. <https://patrickgod.com/sleep-more-code-more>
- Pinterest. (s.f.). *Hola mundo en diversos lenguajes* [Imagen]. <https://www.pinterest.com.mx/pin/857091372823715133/>
- Prince, A. (2020). *El autoaprendizaje como proceso para la construcción de conocimientos en tiempos de pandemia*. <http://portal.amelica.org/ameli/jatsRepo/400/4001720008/4001720008.pdf>
- Rincón Rodríguez, O. O., & Aldana Bautista, L. (2021). Cultura organizacional y su relación con los sistemas de gestión: Una revisión bibliográfica. *Signos, Investigación en Sistemas de Gestión*, 13(2). <https://doi.org/10.15332/24631140.6675>
- solobenja. (2009). *Novedades en Java 6*. <https://solobenja.wordpress.com/2009/12/09/novedades-en-java-6-i/>
- Torres, Y., Camacho, J., Solano, J., & León, L. (2020). *Migración de un monolito a una arquitectura basada en microservicios, caso de estudio sistema "kbus"*. <https://www.dominiodelasciencias.com/ojs/index.php/es/article/view/1193/1886>
- Vasilliy. (2020). *What can we learn from the demise of RxJava?* <https://www.techyourchance.com/rxjava-lessons-learned/>
- Velázquez, P., Flores, B., Ibarra, J., Astorga, M., Aguilar, R., Ramírez, E., & López, G. (2021). *Identificación de áreas de aplicación de arquitecturas de software basadas en modelos, técnicas y herramientas de social media*. <https://scielo.pt/pdf/rist/n42/1646-9895-rist-42-12.pdf>
- Vita, M. (2021). *Plataformas educativas y herramientas digitales para el aprendizaje*. <https://repository.uaeh.edu.mx/revistas/index.php/prepa4/article/view/7593/8211>
- Vives, L. (2021). *Estrategia de autoaprendizaje de programación, sustentada en un modelo de gamificación integral contextualizado*. <https://repositorio.uss.edu.pe/bitstream/handle/20.500.12802/9053/Vives%20Garnique%20Luis%20Alberto.pdf?sequence=1&isAllowed=y>
- Warburton, R. (2015). *Java 8 Lambdas: Pragmatic functional programming: Why did they need to change Java again*. <https://books.google.es/books?hl=es&lr=&id=FKUdAwAAQBAJ&oi=fnd&pg=PR2&dq=functional+programming+java+&ots=5k-Nd6EcVc&sig=rJ4P4NIV9OwWoBMNn3kveommjHo#v=onepage&q=functional%20programming%20java&f=false>