Ciencia Digital

www.cienciadigital.org

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Desarrollo de una guía metodológica para el desarrollo de aplicaciones del lado del servidor

Development of a methodological guide for the development of server-side applications

[1] Byron Gustavo Loarte Cajamarca https://orcid.org/0000-0001-8954-8002
Universidad Escuela Politécnica Nacional, Escuela de Formación de Tecnólogos, Quito, Ecuador,
byron.loarteb@epn.edu.ec

Artículo de Investigación Científica y Tecnológica
Enviado: 13/07/2024
Revisado: 10/08/2024
Aceptado: 26/09/2024
Publicado:05/10/2024
DOI: **https://doi.org/10.33262/cienciadigital.v8i4.3206**

Cítese: Loarte Cajamarca, B. G. (2024). Desarrollo de una guía metodológica para el desarrollo de aplicaciones del lado del servidor. Ciencia Digital, 8(4), 54-74. https://doi.org/10.33262/cienciadigital.v8i4.3206

CIENCIA DIGITAL, es una revista multidisciplinaria, trimestral, que se publicará en soporte electrónico tiene como misión contribuir a la formación de profesionales competentes con visión humanística y crítica que sean capaces de exponer sus resultados investigativos y científicos en la misma medida que se promueva mediante su intervención cambios positivos en la sociedad. https://cienciadigital.org
La revista es editada por la Editorial Ciencia Digital (Editorial de prestigio registrada en la Cámara Ecuatoriana de Libro con No de Afiliación 663) www.celibro.org.ec

Ciencia Digital
ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024
www.cienciadigital.org

Palabras claves:
desarrollo de software, guía metodológica, backend, API RESTFul

Resumen

Introducción. En un contexto global cada vez más digitalizado el desarrollo de aplicaciones del lado del servidor, conocido como *backend*, ha emergido como una competencia fundamental para quienes deseen especializarse en esta área del desarrollo de software. Sin embargo, muchos desarrolladores principiantes enfrentan dificultades significativas al no contar con una orientación clara y estructurada que abarque todo el proceso de desarrollo, tales como la falta de guías comprensivas que detallen los pasos necesarios para crear aplicaciones *backend* que sean robustos, eficientes y escalables. Además, esta carencia puede llevar a tomar decisiones incorrectas que comprometan la calidad del software, su funcionalidad y su capacidad de adaptación a largo plazo. Además, cuestiones como la selección de lenguajes de programación adecuados, la adopción de patrones de arquitectura eficientes, una adecuada elección de base de datos, y la implementación de pruebas y despliegue suelen ser factores críticos que complican el proceso de desarrollo para aquellos sin experiencia previa. **Objetivo**. Desarrollar una guía metodológica que estructure de manera detallada las etapas involucradas en el desarrollo de aplicaciones *backend*, abarcando desde la identificación de la problemática inicial hasta el despliegue en entornos de producción. Esta guía busca proporcionar claridad en cada fase del desarrollo, facilitando la toma de decisiones informadas sobre tecnologías, herramientas, lenguajes y arquitecturas, optimizando la eficiencia y calidad del proceso de desarrollo de software. **Metodología.** La metodología empleada en este trabajo sigue un enfoque deductivo, con un diseño descriptivo y cualitativo, llevándose a cabo una revisión exhaustiva de la literatura especializada en desarrollo de software, metodologías ágiles y tradicionales, así como casos de estudio enfocados en la implementación de proyectos *backend*. A través del análisis de dichos estudios y la experiencia práctica en el campo, se construyó una estructura metodológica sólida que puede ser aplicada en diversos contextos. **Resultados**. Entre los resultados obtenidos en la investigación se confirma que la adopción de una metodología clara y bien definida mejora significativamente el proceso de desarrollo *backend*. Además, la integración de herramientas y *Frameworks* actuales, no solo estandariza el

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

desarrollo, sino que incrementa la productividad de los equipos, reduce los errores y asegura un mejor manejo de la lógica del negocio y la manipulación de datos. Además, la correcta implementación de pruebas unitarias, funcionales y de rendimiento mejora la calidad del software, garantizando su estabilidad y capacidad de escalabilidad. **Conclusión.** Se concluye que esta guía metodológica provee una base sólida para el desarrollo eficiente de aplicaciones *backend*, asegurando una adecuada gestión del ciclo de vida del software. **Área de estudio general:** Informática o Ciencias de la Computación. **Área de estudio específica:** Desarrollo de software. **Tipo de estudio:** Artículo original.

Keywords:
software development, methodological guide, backend, RESTful API

Abstract

Introduction. In an increasingly digital global context, the development of server-side applications, known as backend, has emerged as a fundamental skill for those seeking to specialize in this area of software development. However, many beginner developers face significant challenges due to the lack of clear and structured guidance that encompasses the entire development process, such as comprehensive guides that outline the necessary steps to create robust, efficient, and scalable backend applications. This absence often leads to poor decision-making, which can compromise software quality, functionality, and long-term adaptability. Furthermore, critical aspects such as the selection of appropriate programming languages, the adoption of efficient architectural patterns, the choice of a suitable database, and the implementation of testing and deployment strategies are key factors that complicate the development process, especially for those without prior experience. **Objective.** The aim of this study is to develop a methodological guide that systematically structures the stages involved in backend application development, covering everything from identifying the initial problem to deploying in production environments. This guide seeks to provide clarity in each phase of development, facilitating informed decision-making regarding technologies, tools, languages, and architectures, while optimizing the efficiency and quality of the software development process. **Methodology.** The methodology used in this work follows a deductive approach with a

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

descriptive and qualitative design, conducting a thorough review of specialized literature on software development, both agile and traditional methodologies, as well as case studies focused on backend project implementation. Through the analysis of these studies and practical field experience, a solid methodological framework was built, which can be applied in various contexts. **Results.** The research confirms that the adoption of a clear and well-defined methodology significantly improves the backend development process. Additionally, the integration of modern tools and frameworks not only standardizes development but also enhances team productivity, reduces errors, and ensures better handling of business logic and data manipulation. Moreover, the correct implementation of unit, functional, and performance tests improves software quality, guaranteeing stability and scalability. **Conclusion.** It is concluded that this methodological guide provides a solid foundation for the efficient development of backend applications, ensuring proper management of the software life cycle.

Introducción

En un mundo cada vez más digitalizado, el desarrollo de aplicaciones del lado del servidor, comúnmente conocido como *backend*, se ha consolidado como una competencia esencial para quienes deseen adentrarse en el campo del desarrollo de software. No obstante, iniciar en esta área puede resultar desafiante saber por dónde empezar, y con ello surgen interrogantes fundamentales, tales como: ¿Qué implica el desarrollo de aplicaciones del lado del servidor? ¿Cuáles son las fases que componen el proceso de desarrollo de software? ¿Qué lenguajes de programación predominan y cómo impactan en un proyecto específico? ¿Qué patrones de arquitectura son adecuados de implementar? ¿Qué tipo de base de datos es más conveniente? ¿Qué pruebas deben realizarse? ¿Y qué plataformas son idóneas para la implementación en producción? Sin duda, todos estos cuestionamientos pueden resultar en una tarea tediosa, que sin una adecuada orientación puede desmotivar a los desarrolladores principiantes y llevarlos a tomar decisiones precipitadas que comprometan la calidad y la escalabilidad del proyecto, o incluso a su eventual abandono. Por ello, es fundamental contar con una guía bien estructurada que no solo aborde estas inquietudes, sino que también proporcione un camino claro y sistemático para dominar el desarrollo *backend*, abarcando desde la comprensión de los

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

fundamentos hasta la implementación de soluciones robustas y eficientes en entornos reales (Coppola, 2023).

Previo a detallar los pasos de la presente guía metodológica, es imperativo realizar una caracterización precisa de los conceptos de diseño web y desarrollo web, ya que, aunque a primera vista puedan parecer similares, poseen distinciones importantes. Por un lado, el diseño web, también conocido como desarrollo *frontend* o desarrollo de aplicaciones del lado del cliente, se centra en aspectos vinculados a la apariencia y percepción de un sitio web, haciendo hincapié en la estética y la experiencia del usuario, con el objetivo de crear interfaces atractivas, interactivas, funcionales, fáciles de usar y sobre todo que se ajusten al objetivo para el cual fue creado y por ello está íntimamente relacionado conceptos tales como la experiencia del usuario (UX) y la interfaz de usuario (IU). Por otro lado, el desarrollo web comprende el proceso integral de la creación ya sea de sitios, aplicaciones o sistemas web que pueden operar tanto en Internet como en redes internas (Intranet). Además, incluye todo lo relativo a la lógica del negocio, los lenguajes de programación, la gestión de bases de datos, y asegura que las aplicaciones respondan adecuadamente a las interacciones del usuario, razón por la cual suele denominarse la parte no visible o comúnmente denominado desarrollo *backend* o desarrollo de aplicaciones del lado del servidor (Pérez et al., 2021).

A continuación, se presenta una tabla comparativa que resalta las principales diferencias entre diseño web y desarrollo web.

Tabla 1

Diferencias entre diseño y desarrollo web

Características	Diseño web	Desarrollo web
Denominación	Comúnmente denominado *frontend* o aplicaciones del lado del cliente	Comúnmente denominado *backend* o aplicaciones del lado del servidor
Enfoque principal	Se enfoca principalmente en apariencia visual (UI) y experiencia del usuario (UX)	Se enfoca principalmente en la lógica del negocio, la funcionalidad y la integración de datos
Tecnologías comunes	HTML, CSS, JavaScript y *Frameworks frontend*	PHP, Python, Ruby, Java, Node.js, *Frameworks backend* y Bases de datos (MySQL o MongoDB)
APIs	Consumir APIs para integrar datos y funcionalidades en las interfaces de usuario	Diseñar, desarrollar y mantener APIs para la comunicación entre el *frontend* y el *backend*, o con servicios externos

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

A continuación, se presenta de una forma sistematizada todos los pasos que requiere un desarrollador *backend* para el desarrollo de aplicaciones del lado del servidor.

Problemática

La problemática o planteamiento del problema constituye una descripción precisa y clara de un problema o desafío que necesita ser abordado el cual constituye un paso fundamental en el proceso de investigación científica, de esta manera una formulación exacta del problema no solo refleja una comprensión profunda del tema en cuestión, sino que también establece una base sólida para el desarrollo de soluciones efectivas (Torres-Rodríguez & Monroy-Muñoz, 2020).

La selección y definición del tema, la formulación de los objetivos, así como la delimitación del problema, son pasos cruciales que orientan no solo la dirección del estudio o en este caso al desarrollo de un producto software, sino que también mejora significativamente la relevancia y el impacto de los resultados obtenidos (Pamplona, 2022).

La manera en que se plantea un problema puede variar según la perspectiva del sujeto, ya sea estudiante, profesional o emprendedor, pero comprender cómo redactar un planteamiento del problema de manera efectiva es fundamental y para ello, se deben seguir ciertas pautas al elaborar la descripción del problema:

- **Contextualización:** Se debe proporcionar una descripción detallada del contexto en el cual surge el problema, lo que facilitará la comprensión de su relevancia y el impacto en diversos aspectos del entorno.
- **Justificación:** Se debe explicar por qué el problema es significativo y por qué requiere atención.
- **Impacto:** Se debe detallar cómo el problema afecta a las partes involucradas y el cual puede incluir consideraciones sobre aspectos sociales, económicos, técnicos, entre otros.
- **Interesados:** Se debe identificar quiénes son los afectados por el problema y quiénes podrían beneficiarse de su resolución.
- **Objetivos Claros:** Se debe definir los objetivos específicos que se pretenden alcanzar con la resolución del problema, asegurando que estos objetivos sean alcanzables y medibles.

En proyectos de desarrollo de software un planteamiento bien formulado clarifica el problema, ofrece el contexto necesario desde la situación actual y actúa como una guía para el desarrollo e implementación de soluciones efectivas.

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Alcance

En el desarrollo de productos software el alcance se refiere al objetivo final a donde se espera llegar o, dicho de otra forma, describe una meta por alcanzar e indica hasta dónde va el proyecto. Por ello, su definición reviste una importancia fundamental ya que al proyectar adecuadamente el alcance dependerá en gran medida los esfuerzos y recursos que se tengan que realizar para alcanzar los objetivos establecidos (Hernández-Sampieri et al., 2014).

Por otra parte, al momento de definir el alcance es fundamental tener en cuenta los requisitos del proyecto, incluyendo el tiempo, las especificaciones, los recursos y las restricciones que deben cumplirse para satisfacer a las partes interesadas. Esto implica la definición clara de roles, la clasificación de requisitos según su relevancia, urgencia y viabilidad, así como su categorización en requisitos funcionales, no funcionales o técnicos, entre otros aspectos. Por lo tanto, comprender y gestionar de manera efectiva el alcance y las limitaciones del proyecto es vital para asegurar su éxito en el desarrollo del producto software (Aguirre & Gil, 2021).

A continuación, se detallan ciertas pautas que se debe tener en cuenta en la elaboración del alcance.

- **Identificación y comprensión de necesidades:** Es vital identificar y comprender en profundidad las necesidades y expectativas de todas las partes interesadas, así como los objetivos esperados, los problemas a resolver, los recursos disponibles y los entregables proyectados.
- **Definición de objetivos:** Los objetivos del proyecto deben ser claros, específicos, y alcanzables, proporcionando una orientación precisa para su desarrollo. Para este propósito, puede emplearse la metodología S.M.A.R.T., que asegura que los objetivos sean específicos, medibles, alcanzables, relevantes y limitados en el tiempo.
- **Descripción de actividades:** A partir de una serie de requerimientos se debe describir cuáles son las acciones que se deben seguir para cumplir a cabalidad con los mismos, el cual puede incluir la clasificación de requisitos funcionales, no funcionales, así como la asignación de roles y responsabilidades.
- **Definición de limitaciones:** Detallar lo que estará incluido y excluido del proyecto es esencial para evitar confusiones, malentendidos y sobre todo evita la inclusión de tareas no planificadas.

Definir el alcance de manera clara y detallada permite establecer una visión común entre todos los miembros del equipo y las partes interesadas. Además, facilita la comprensión y reduce significativamente los malentendidos y conflictos durante la ejecución del proyecto, garantizando así un desarrollo más cohesionado y eficiente.

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Marco Teórico

El marco teórico, tiene como función principal proporcionar una estructura lógica y ordenada de la teoría y antecedentes que fundamentan el proyecto a desarrollar. Similar a cómo antes de explorar un territorio desconocido uno necesita una brújula, un mapa detallado y ciertas indicaciones sobre lo que podría encontrarse, en una investigación o proyecto, el marco teórico actúa como esa guía indispensable, pues proporciona la orientación, el contexto y el conocimiento necesario para abordar el problema de manera sistemática y fundamentada (Zamorano, 2013).

Además, el marco teórico ayuda a definir todos los términos y conceptos clave que están relacionados con el tema y a explicar por qué son relevantes y pertinentes para el proyecto a desarrollar, lo que asegura que su significado sea claro y consistente para otros lectores o para los miembros del equipo de trabajo (Universidad Privada del Norte, 2022).

No existe una receta única para elaborar un marco teórico, pero es posible seguir ciertas pautas para su correcta redacción:

- Realizar una revisión exhaustiva de la bibliografía (incluyendo fuentes primarias y secundarias) y seleccionar únicamente lo que sea relevante para el proyecto.
- Identificar y citar las bases legales, si son necesarias para el proyecto.
- Organizar los conceptos de forma jerárquica y lógica, facilitando su comprensión.
- Evitar incluir información irrelevante, enfocándose exclusivamente en datos que contribuyan al avance del conocimiento.
- Redactar los conceptos con claridad para evitar interpretaciones incorrectas.
- El marco teórico no debe dividirse en capítulos; en cambio, cada sección debe ser presentada con el título correspondiente.

Estas pautas no solo contribuyen a la construcción de un marco teórico sólido, sino que también garantiza su coherencia y rigor académico.

Metodología

Para la presente guía metodológica se hace uso del estudio de casos, el cual es un tipo de metodología de investigación que se utiliza para analizar en profundidad un fenómeno o situación particular en un contexto específico. Este enfoque se emplea con el objetivo de comprender, de manera detallada, las complejidades de un caso único o un pequeño grupo de casos, y puede utilizarse tanto en estudios cualitativos como cuantitativos (Loarte & Maldonado, 2019).

Una vez identificado la problemática, definidos los objetivos, establecido el alcance y desarrollado el marco teórico, resulta indispensable avanzar hacia la selección de la metodología que regirá el desarrollo del proyecto. En el ámbito del desarrollo de software,

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

la selección de una metodología adecuada es de vital importancia, ya que no solo debe abordar los aspectos técnicos, sino también integrar procesos que garanticen la calidad, escalabilidad y el mantenimiento a largo plazo del producto.

Resultados

Una metodología de desarrollo de software constituye un marco de trabajo que abraca un conjunto de prácticas y estrategias las cuales están destinadas a la creación de soluciones software. Además, cabe destacar que la diversidad de metodologías disponibles en el panorama actual implica que la selección de la más adecuada responde a las particularidades de cada equipo de desarrollo, con el propósito de optimizar la organización y la eficiencia en la ejecución del proyecto. Por consiguiente, la elección metodológica debe ser cuidadosamente evaluada, considerando tanto enfoques ágiles como tradicionales, los cuales serán analizados a continuación (Morales-Carrillo et al., 2022).

Las metodologías tradicionales o en cascada

Como su nombre lo indica estas metodologías se han utilizado tradicionalmente a lo largo del tiempo y se caracterizan por imponer una rigurosa disciplina en el proceso de desarrollo de software, con el fin de hacerlo más predecible. Esto implica la adopción de un enfoque secuencial en el que las fases se ejecutan en una única dirección, sin posibilidad de retroceso. Asimismo, la recopilación de requisitos se realiza una única vez al inicio del proyecto, constituyéndose en un proceso de extrema rigurosidad y relevancia ya que de ella dependerá de todos los recursos que se vayan a emplear en el proyecto y de la estructuración de las fases subsecuentes, que incluyen la planificación, ejecución, monitoreo y cierre del proyecto (Smartsheet LATAM, 2022).

Las metodologías ágiles

Se fundamentan en ciclos iterativos en los que los proyectos se descomponen en pequeñas tareas o fases de corta duración, conocidas como *Sprints*. A diferencia del enfoque tradicional, en este marco metodológico se invierte menos tiempo en la planificación y priorización previa, dado que su estructura es más flexible frente a cambios en los requerimientos iniciales los cuales pueden evolucionar a lo largo del proyecto, promoviendo de esta manera una retroalimentación continua hacia los usuarios finales. Además, el objetivo de cada iteración es entregar un producto funcional, lo que convierte a las metodologías ágiles en una opción particularmente adecuada para proyectos en los que el cliente no tiene claridad absoluta sobre el resultado final, requiere tiempos de entrega más ágiles y desea involucrarse activamente en el proceso de diseño y desarrollo (Bautista-Villegas, 2022).

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

A continuación, se presenta una tabla comparativa que resalta las principales diferencias entre las metodologías ágiles y tradicionales.

Tabla 2

Diferencias entre metodologías tradicionales y metodologías ágiles

Características	Metodologías tradicionales	Metodologías ágiles
Tipos	Cascada, Modelo en V y Modelo Incremental	Scrum, Kanban, Lean y XP (Extreme *Programming*)
Estructura organizativa	Lineal	Iterativa
Escala de proyectos	Grandes (pero también aplicable a medianos)	Pequeños, medianos y escalables a proyectos grandes
Requisitos	Bien definidos antes de empezar, no flexibles	Dinámicos, susceptibles a cambios durante el proyecto
Implicación del cliente	Baja	Alta
Modelo de desarrollo	Ciclo de vida secuencial	Entrega evolutiva e incremental
Planificación	Se planifica todo con gran detalle al inicio	Planificación ajustada de Sprint en Sprint
Duración de los ciclos	Ciclos largos, con entregas al final de cada fase.	Ciclos cortos (2-4 semanas por Sprint) con

Es evidente que la selección de la metodología debe estar respaldada por un análisis riguroso de los requisitos del proyecto, la naturaleza del producto a desarrollar y las expectativas de los *stakeholders*, asegurando así, que la metodología escogida esté alineada con los objetivos del proyecto, garantice el cumplimiento de los estándares de calidad previamente establecidos y sobre todo la consecución exitosa del proyecto.

Patrón de arquitectura

Tras la selección de la metodología de desarrollo y el inicio de la fase de codificación, es crucial optar por un patrón de arquitectura adecuado, el cual permita que el desarrollo del software sea de calidad, seguro, escalable y fácilmente mantenible; contribuyendo así a la sostenibilidad y eficiencia del software a lo largo de su ciclo de vida. En este contexto, un patrón de arquitectura se define como una solución general, reutilizable y probada para abordar problemas recurrentes en la ingeniería de software, en sí, se trata de un marco de referencia que guía a los equipos de desarrollo en la construcción y diseño de un sistema determinado (Giraldo et al., 2021).

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

A pesar de lo expuesto previamente, no debe inferirse que exista un único camino para implementar un patrón de arquitectura pues en realidad, existen diversos patrones arquitectónicos, entre los que se incluyen el patrón en capas, el monolítico, los microservicios y los basados en eventos, entre otros (Loarte & Maldonado, 2019). Sin embargo, el patrón que ha alcanzado mayor popularidad y uso es el Modelo-Vista-Controlador (MVC), un enfoque arquitectónico de tres capas que se detalla a continuación:

- **Modelo:** Esta capa gestiona los datos, la lógica de negocio y las reglas que deben cumplirse en el sistema.
- **Vista:** Encargada de la presentación de los datos, esta capa se comunica con el usuario mediante las interacciones.
- **Controlador:** Actúa como intermediario entre el modelo y la vista, procesando las solicitudes del usuario, realizando las operaciones correspondientes en el modelo y actualizando la vista.

Gracias a su flexibilidad, el patrón MVC se utiliza ampliamente en una variedad de sistemas, desde aplicaciones básicas hasta complejos sistemas empresariales, simplificando la administración y escalabilidad del código de una manera eficiente y sostenible.

Base de datos

Otro aspecto fundamental para considerar es la gestión de la información, para lo cual existen dos grandes tipos de bases de datos: relacionales (SQL) y no relacionales (NoSQL). De esta manera, las bases de datos relacionales organizan los datos en tablas, columnas y establecen relaciones entre ellas, priorizando la integridad de los datos y garantizando sobre todo la escalabilidad. En cambio, las bases de datos NoSQL no requieren relaciones entre los datos y, a menudo, se clasifican en función de cómo almacenan la información, lo que facilita que la ejecución de consultas más sencillas y directas (Loarte, 2022).

En el desarrollo de aplicaciones del lado del servidor, una vez definida la base de datos a utilizar, es crucial implementar un Mapeo Objeto-Relacional (ORM). El ORM es una técnica de programación que permite mapear las tablas de una base de datos relacional o las colecciones de una base de datos no relacional en clases denominadas entidades. Además, esta técnica elimina la necesidad de escribir consultas SQL manualmente, ya que el ORM se encarga de generarlas, independientemente del motor de base de datos que se esté utilizado reduciendo la complejidad del código e incrementando la productividad en el desarrollo de software.

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Por último, la elección del ORM varía según el lenguaje de programación empleado, y entre los más utilizados se encuentran:

- Hibernate para-Java.
- SQLAlchemy o Django ORM para Python.
- Entity Framework para-C#.
- ActiveRecord para Ruby.
- Sequelize, Prisma, Mongoose o TypeORM para JavaScript/Node.js.
- Eloquent para PHP.
- GORM para Golang.

Herramientas de desarrollo

En el ámbito del desarrollo de software, se dispone de una amplia gama de herramientas, librerías y lenguajes de programación, cada uno destinado a facilitar y optimizar el proceso de desarrollo. La selección de estas herramientas no solo está influenciada por criterios técnicos, sino que también depende, en muchos casos, del nivel de experiencia y la familiaridad del desarrollador con dichas tecnologías. Además, los equipos de desarrollo emplean estos recursos con el propósito de superar los retos inherentes al momento de escribir código, probar programas, implementar aplicaciones y supervisar las versiones de producción ya que el uso adecuado de estas herramientas no solo acelera la codificación, sino que también permite identificar y corregir errores, optimizar los flujos de trabajo y mejorar diversos aspectos del ciclo de desarrollo (Rojas, 2020).

Entre los principales beneficios derivados de una elección acertada de herramientas de desarrollo destacan:

- Mejora en la calidad del código.
- Reducción de errores en la codificación.
- Disminución del tiempo de desarrollo.
- Mejora de la colaboración entre equipos.
- Reducción de costos operativos.

Las herramientas de desarrollo de software son, por tanto, aplicaciones diseñadas específicamente para facilitar y optimizar las distintas fases del ciclo de vida del software, como su creación, prueba y mantenimiento. Además, estas herramientas resultan indispensables para incrementar la productividad de los desarrolladores, asegurar la calidad del código generado y gestionar de manera eficiente los proyectos de software, contribuyendo así a la sostenibilidad de estos.

Por otro lado, un Framework constituye un entorno de trabajo que proporciona a los desarrolladores un conjunto predefinido de herramientas y directrices para la creación de

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

sistemas software. Al contar con una estructura previamente establecida, los *Frameworks* permiten una serie de ventajas sustanciales, tales como la reutilización de código y componentes, la simplificación del desarrollo, la integración de patones de arquitectura, la incorporación de funcionalidades integradas y la compatibilidad con bibliotecas externas, entre otras más. Actualmente, existen diversos *Frameworks* diseñados para cada lenguaje de programación. Ejemplos notables incluyen Django y Flask para Python; React, Angular y Vue.js para JavaScript; Spring para Java; Ruby on Rails para Ruby; Laravel para PHP; y ASP.NET para C#, entre otros.

El uso de *Frameworks* no solo facilita la implementación de soluciones robustas y escalables, sino que también contribuye a la adopción de buenas prácticas en el desarrollo de software, lo cual es fundamental para garantizar el éxito a largo plazo de cualquier proyecto tecnológico (Loarte, 2022).

Codificación

La fase de codificación en el ciclo de vida del desarrollo de software representa un momento crucial en el que los requisitos previamente definidos por el cliente son transformados en código ejecutable. Esta transformación se lleva a cabo utilizando una metodología de desarrollo específica, junto con un patrón arquitectónico, un lenguaje de programación adecuado y un Framework en particular para que los desarrolladores se encarguen de escribir el código fuente y que el producto final cumpla con las expectativas establecidas en cada uno de los entregables.

Cabe destacar que, durante el proceso de codificación los desarrolladores no solo implementan el código, sino que también realizan pruebas unitarias con el objetivo de garantizar que cada componente del software funcione de manera correcta y autónoma antes de proceder con la integración de estos componentes en otros módulos. Además, este enfoque permite identificar y resolver errores en una fase temprana del desarrollo, optimizando así la calidad del software antes de su integración completa y posterior despliegue en entornos de producción. La codificación, por lo tanto, no es solo la traducción técnica de los requisitos funcionales, sino también un proceso riguroso de validación y aseguramiento de la calidad, que sienta las bases para el éxito de las fases posteriores del ciclo de desarrollo del software.

Pruebas

El proceso de pruebas de software es esencial dentro del ciclo de vida del desarrollo, ya que cumple una doble función: por un lado, permite asegurar la calidad y la funcionalidad de cualquier producto en desarrollo, y por otro, proporciona la mayor garantía de que dicho producto está libre de defectos y cumple con el comportamiento esperado. En este sentido, las pruebas de software constituyen el método más efectivo para verificar el

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

correcto funcionamiento del producto a lo largo de todas las etapas del desarrollo ya se utilizando pruebas manuales y las automatizadas (Caicedo, 2023).

Las pruebas manuales requieren la intervención directa de una persona que actúa como evaluador. Además, este análisis puede basarse en casos de prueba previamente definidos o en una exploración intuitiva del software, con el objetivo de identificar problemas imprevistos. Esta modalidad es especialmente valiosa en las pruebas de usabilidad, donde la evaluación de la interfaz de usuario, la interacción con la aplicación y el uso de las API requiere una perspectiva humana, que resulta crucial para identificar aspectos que podrían pasar desapercibidos en pruebas automatizadas.

En contraposición, las pruebas automatizadas se valen de scripts o herramientas especializadas para ejecutar los casos de prueba y verificar automáticamente los resultados esperados. Además, la automatización se traduce en un ahorro significativo de tiempo y esfuerzo, especialmente en proyectos grandes, permitiendo la ejecución simultánea de múltiples pruebas de forma consistente, lo que incrementa la eficiencia en comparación con las pruebas manuales (Camacho, 2024).

Existen diversos tipos de pruebas de software, cada uno con características específicas que se detallan a continuación en la siguiente tabla.

Tabla 3

Tipos de pruebas de software

Tipo	Característica	Ejemplo
Unitarias	Comprueban que cada una de las piezas o unidades más pequeñas del software (fragmentos de código) en el que se está trabajando funcione correctamente. Además, estas pruebas se aplican de manera individual y son las primeras que deben realizarse durante todo el proceso de desarrollo.	Probar la función que registra a un usuario y comprobar que el registro sea correcto.
Integración	Se encargan de verificar que los distintos módulos o servicios utilizados por el software funcionen bien en conjunto.	Verificar que el servicio de envío de correos electrónicos interactúa correctamente con el módulo de registro de usuarios.
Funcionales	Estas pruebas comprueban que las funciones del software funcionen adecuadamente emulando escenarios de negocio, basándose en los requisitos funcionales.	Comprobar que el formulario de registro permite crear una cuenta cuando todos los campos son válidos y que los datos se almacenan en la base de datos.

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Tabla 3

Tipos de pruebas de software (continuación)

Tipo	Característica	Ejemplo
Rendimiento	Ayudan a evaluar el rendimiento del software con una carga de trabajo determinada. Además, ayudan a medir la fiabilidad, la velocidad, la escalabilidad y la capacidad de respuesta.	Medir el tiempo de respuesta del software cuando 500 usuarios intentan acceder simultáneamente al módulo de registro.
Estrés	Son esenciales para garantizar que un software sea confiable bajo condiciones extremas, permitiendo anticiparse a fallos críticos y mejorar la experiencia del usuario en situaciones de alta carga.	Medir el tiempo de respuesta del software cuando 10,000 usuarios intentan acceder simultáneamente al módulo de registro durante una venta especial.
Aceptación	Se encargan de verificar si el software satisface todos los requisitos del negocio y funcione según lo previsto. Además, la aprobación por lo general la realizan los *stakeholders*.	Validar que el flujo de registro de usuarios en el software cumple con los requisitos del cliente final y los objetivos del negocio.

Despliegue

Tras finalizar la etapa de codificación y pruebas, es crucial proceder con el despliegue del producto software hacia un entorno de producción el cual es un proceso que involucra una serie de actividades necesarias para asegurar que el software esté preparado para su utilización, ya sea en dispositivos o servidor. Dentro de estas actividades se incluyen la liberación, instalación, pruebas, despliegue y monitoreo del software (Almora et al., 2022).

El despliegue puede realizarse internamente en una Intranet, garantizando el acceso exclusivo al personal de la organización, o bien, puede optarse por un despliegue externo a través de Internet utilizando tres modelos principales de servicios en la nube que son Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS). Cabe destacar que muchas organizaciones combinan estos modelos con soluciones de TI tradicionales, especialmente las de mayor tamaño, que tienden a utilizar más de uno de estos enfoques (Buenning, 2024).

Por otra parte, la estrategia de despliegue varía según su naturaleza, sin embargo, existen fases comunes dentro del proceso, descritas a continuación:

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

- **Planificación:** Es esencial diseñar un plan exhaustivo que detalle los recursos necesarios, como hardware, software y personal, para llevar a cabo el despliegue.
- **Diseño de la estrategia:** Una vez planificado, se debe definir la metodología más eficaz para implementar dicho plan.
- **Empaquetado del software:** El software debe ser preparado en un formato adecuado, ya sea mediante un instalador o mediante una imagen de contenedor, para facilitar su despliegue.
- **Simulación:** Se deben llevar a cabo simulaciones que imiten con precisión el entorno de la organización, a fin de detectar posibles problemas antes del despliegue final, asegurando así la plena operatividad del software.
- **Despliegue:** Se procede con la implementación en el entorno de producción, siguiendo rigurosamente los procedimientos de seguridad y cumplimiento normativo.
- **Monitoreo y mantenimiento:** Es fundamental supervisar el rendimiento del software en producción y realizar mantenimientos preventivos o correctivos según sea necesario para garantizar su funcionamiento óptimo.

Por último, el éxito de un despliegue sin contratiempos es esencial para el correcto funcionamiento de una organización, el cual es un proceso que puede gestionarse de forma manual o mediante la automatización utilizando enfoques de integración y despliegue continuo (CI/CD).

Conclusiones

- El desarrollo de aplicaciones del lado del servidor, conocido como *backend*, desempeña un rol crucial en asegurar tanto la funcionalidad como la escalabilidad de productos software complejos. En este sentido, la presente guía metodológica detalla de una forma minuciosa todos los pasos necesarios para llevar a cabo dicho desarrollo, desde la identificación del problema hasta su despliegue en producción.
- Si bien los roles del *frontend y backend* se interrelacionan en la construcción de productos software, sus enfoques, tecnologías y objetivos presentan diferencias sustanciales. Por esta razón, es importante tener en cuenta que el diseño web se orienta hacia la experiencia del usuario y la apariencia visual, mientras que el desarrollo *backend* se enfoca en la lógica de negocio y la gestión de datos, evidenciando así la complementariedad de habilidades requeridas para lograr un desarrollo web integral.
- La selección de una metodología de desarrollo adecuada es fundamental para el éxito de un proyecto software ya que, por una parte, las metodologías ágiles con su carácter iterativo y adaptable se ajustan mejor a proyectos con requisitos en constante evolución. Por el contrario, los enfoques tradicionales resultan más

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

apropiados en aquellos proyectos donde las especificaciones están claramente definidas y permanecen estables durante todo el ciclo de desarrollo.

- La elección de un patrón de arquitectura apropiado, como el Modelo-Vista-Controlador (MVC), es determinante para la creación de productos software escalables y sostenibles. Este patrón no solo facilita la organización del código mediante la separación de responsabilidades entre la lógica de negocio, la presentación de datos y la interacción con los usuarios, sino que también asegura una mayor facilidad de mantenimiento a largo plazo.
- La correcta elección de la base de datos ya sea relacional (SQL) o no relacional (NoSQL), es vital para que el sistema software pueda manejar eficientemente grandes volúmenes de información. Además, el uso de un Mapeo Objeto-Relacional (ORM) en la integración de la base de datos optimiza las operaciones de acceso y manipulación de datos, lo que incrementa la coherencia y simplifica la complejidad del código.
- La adopción de un Framework adecuado no solo estandariza el desarrollo, sino que además mejora significativamente el proceso de codificación al proporcionar componentes predefinidos y promover buenas prácticas. Además, *Frameworks* como Django, *Flask* o Spring son ejemplos que incrementan la productividad y reducen el margen de error en el desarrollo de *backend*, acelerando los ciclos de desarrollo sin comprometer la calidad.
- Las herramientas de desarrollo influyen directamente en la eficiencia del ciclo de vida del software. En ese sentido, una buena elección de herramientas especializadas para la automatización de pruebas, la gestión de versiones y la integración de *Frameworks* mejora no solo la calidad del código, sino también la colaboración entre equipos, reduciendo así el tiempo de desarrollo y asegurando la entrega de productos con altos estándares.
- La fase de codificación debe ser complementada con la ejecución de pruebas unitarias que garanticen el correcto funcionamiento de cada componente individual del software. Además, la integración de estas pruebas en el ciclo de desarrollo permite la detección temprana de errores, optimizando la calidad general del software antes de su integración final.
- Las pruebas de software, tanto manuales como automatizadas, son fundamentales para validar la funcionalidad y calidad de un software. Además, la implementación de pruebas unitarias, de integración y de rendimiento, permite la identificación y corrección de errores en fases tempranas, lo que mejora la estabilidad y confiabilidad del software antes de su implementación en producción.
- Finalmente, el despliegue de un software en un entorno de producción exige una planificación detallada que garantice su correcto funcionamiento y minimice interrupciones. Además, el uso de herramientas de integración y despliegue

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

continuo (CI/CD) automatiza el proceso, asegurando una implementación más rápida, controlada y con un menor riesgo de fallos, lo que es fundamental para la sostenibilidad del producto final.

Conflicto de intereses

Los autores deben declarar si existe o no conflicto de intereses en relación con el artículo presentado.

Referencias Bibliográficas

Aguirre, M., & Gil, E. (2021). *La guía completa para entender y definir el alcance de un proyecto en 5 pasos y con un ejemplo.* appvizer. https://www.appvizer.es/revista/organizacion-planificacion/gestion-proyectos/alcance-de-un-proyecto

Almora Gálvez, Y., García Rodriguez, A., Gómez Perdomo, Y., & León de la O, D. (2022). Procedimiento para el despliegue de software de gestión. *Revista Cubana de Ciencias Informáticas, 16*(3), 35-50. http://scielo.sld.cu/scielo.php?pid=S2227-18992022000300035&script=sci_arttext

Bautista-Villegas, E. (2022). Metodologías agiles XP y Scrum, empleadas para el desarrollo de páginas web, bajo MVC, con lenguaje PHP y framework Laravel. *Revista Amazonía Digital, 1*(1), e168-e168. https://revistas.unamad.edu.pe/index.php/rad/article/view/168

Buenning, M. (2024). *Guía del proceso de despliegue de software para 2024*. Ninjaone. https://www.ninjaone.com/es/blog/proceso-de-software-deployment/#:~:text=%C2%BFQu%C3%A9%20es%20el%20despliegue%20de,conoce%20como%20despliegue%20de%20aplicaciones.

Caicedo Goyes, F. L. (2023). Mejora de la calidad del software a través de la integración y entrega continua. *Revista Odigos*, *4*(2), 45–55. https://doi.org/10.35290/ro.v4n2.2023.899

Camacho, R. (2024). *Guía de metodologías de prueba de software: una descripción general de alto nivel.* Parasoft. https://es.parasoft.com/blog/software-testing-methodologies-guide-a-high-level-overview/

Coppola, M. (2023). *Desarrollo web: qué es, etapas y principales lenguajes*. Hubspot: https://blog.hubspot.es/website/que-es-desarrollo-web

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Giraldo Mejía, J., Vargas Agudelo, F., & Garzón Gil, K. (2021). Marco de trabajo para seleccionar un patrón arquitectónico en el desarrollo de software. *Revista Ibérica de Sistemas e Tecnologías - RISTI*, E43, 568-581. https://dspace.tdea.edu.co/handle/tdea/2670

Hernández-Sampieri, R., Fernández-Collado, C., & Baptista-Lucio, P. (2014). *Metodología de la Investigación* (6ta edición). Editorial MacGrawHill. https://es.slideshare.net/slideshow/metodologa-de-la-investigacin-sexta-edicinpdf/261930050

Loarte Cajamarca, B. G. (2022). Desarrollo de un backend para la gestión del sistema penitenciario del Ecuador. *Conciencia Digital*, *5*(3.2), 47-66. https://doi.org/10.33262/concienciadigital.v5i3.2.2319

Loarte Cajamarca, B. G., & Maldonado Soliz, I. F. (2019). Desarrollo de una aplicación web y móvil en tiempo real, una evolución de las aplicaciones actuales. *Ciencia Digital*, *3*(1), 201-216. https://doi.org/10.33262/cienciadigital.v3i1.282

Morales-Carrillo, J., Cedeño-Valarezo, L., Bravo, J., & Calderón, J. (2022). Metodologías de desarrollo de software y su ámbito de aplicación: Una revisión sistemática. *Revista Ibérica de Sistemas e Tecnologias de Informação*, *E47*, 29-45. https://www.proquest.com/docview/2648273778?pq-origsite=gscholar&fromopenview=true

Pamplona, F. (2022). *¿Cuál es el planteamiento del problema y cómo debe enmarcarse?* Mindthegraph. https://mindthegraph.com/blog/es/declaracion-del-problema-documento-de-investigacion/

Pérez Ibarra, S., Quispe, J., Mullicundo, F., & Lamas, D. (2021). *Herramientas y tecnologías para el desarrollo web desde el FrontEnd al BackEnd* [Congreso XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja), 347-350. https://sedici.unlp.edu.ar/handle/10915/120476

Rojas, E. (2020). Machine Learning: análisis de lenguajes de programación y herramientas para desarrollo. *Revista Ibérica de Sistemas e Tecnologias de Informação*, *E28*, 586-599. https://www.proquest.com/docview/2388304894?pq-origsite=gscholar&fromopenview=true&sourcetype=Scholarly%20Journals

Smartsheet LATAM. (2022). *Diferencias entre metodologías ágiles y tradicionales: ventajas y desventajas.* Linkedin. https://www.linkedin.com/pulse/diferencias-entre-metodolog%C3%ADas-%C3%A1giles-y-tradicionales-ventajas-/

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024

Torres-Rodríguez, A. A., & Monroy-Muñoz, J. I. (2020). El problema de la definición del problema de investigación. *Boletín Científico de la Escuela Superior Atotonilco de Tula*, *7*(13), 10-15. https://doi.org/10.29057/esat.v7i13.5265

Universidad Privada del Norte. (2022). *Descubre qué es el marco teórico, estructura, función y ejemplos.* https://blogs.upn.edu.pe/estudios-generales/2022/07/14/marco-teorico/

Zamorano García, J. (2013). El marco teórico. *Vida Científica Boletín Científico de la Escuela Preparatoria No. 4*, *1*(2). https://repository.uaeh.edu.mx/revistas/index.php/prepa4/article/view/1808

ISSN: **2602-8085**
Vol. 8 No. 4, pp. 54 – 74, octubre – diciembre 2024