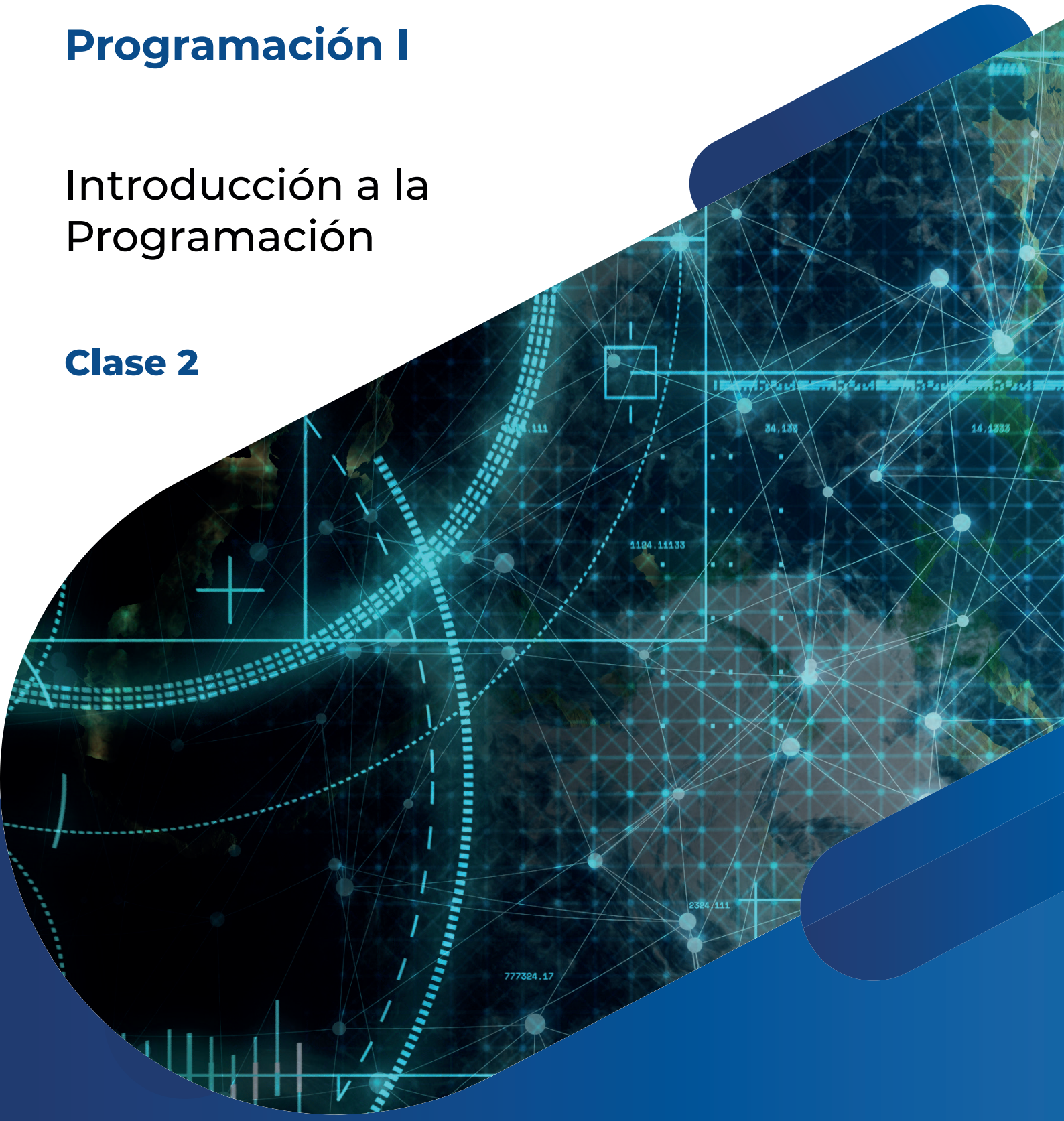
Programación I

Introducción a la Programación

Clase 2



Ingeniería en ciberseguridad

La excelencia no se improvisa

1. INTRODUCCIÓN DE LA CLASE

En esta segunda clase, continuamos nuestro viaje por el mundo de la programación, adentrándonos en la forma en que se organiza el proceso de desarrollo de software de manera sistemática. A lo largo de esta sesión, revisaremos las fases cruciales que conforman dicho desarrollo (análisis, diseño, codificación y prueba), poniendo el foco en cómo cada etapa colabora para minimizar errores, optimizar recursos y fortalecer la calidad de las aplicaciones. Además, incorporaremos herramientas esenciales para la planificación, tales como el pseudocódigo y los diagramas de flujo, que facilitan la elaboración de soluciones claras y escalables. Para comprender la relevancia de estas herramientas, imaginemos a dos estudiantes que discuten la necesidad de una buena preparación antes de programar: el primero insiste en que, sin un problema definido y un método de abordaje sólido, es como armar un rompecabezas sin contar con la imagen de referencia. El segundo estudiante respalda esta idea, enfatizando que pseudocódigo y diagramas de flujo resultan indispensables para descomponer las tareas y visualizar el proceso de manera ordenada.

El profesor subraya la trascendencia de entender cada fase del desarrollo para crear programas que no solo funcionen, sino que sean robustos y eficientes. Durante la clase, exploraremos cómo el análisis permite recopilar requisitos y delimitar el problema; de qué manera el diseño se enfoca en estructurar la solución antes de escribir una sola línea de código; por qué la codificación, basada en un plan previo, reduce el margen de error; y cómo la prueba verifica la funcionalidad y la confiabilidad del producto final. Asimismo, reflexionaremos acerca de la influencia de la planificación en la disminución de costos y tiempos de desarrollo, evitando retrabajos o decisiones improvisadas que perjudiquen el avance. Por último, se hará énfasis en la adopción de buenas prácticas que promuevan el trabajo en equipo de forma coordinada, garantizando que cada integrante entienda su papel y los objetivos comunes. Con estas herramientas y perspectivas, la clase busca demostrar que la programación no es únicamente escribir instrucciones para una computadora, sino concebir un proceso integral donde cada fase aporta un valor específico, asegurando resultados de calidad y favoreciendo la colaboración entre todos los involucrados en el proyecto. La planificación adecuada es la base del éxito en cualquier proyecto de software.

Clase 2: Introducción a la Programación (Parte 2)

Reto # 1

Contenido de la Clase:

2. **Tema: Introducción a la Programación**

En esta sección, profundizaremos en dos aspectos cruciales que resultan determinantes para la planificación y el desarrollo de software. En primer lugar, analizaremos las fases que componen el ciclo de vida de un proyecto de software —concretamente, el análisis, el diseño, la codificación y la prueba—, explicando en detalle cada una de ellas y subrayando su relevancia para la creación de soluciones que no solo sean funcionales, sino también robustas y eficientes. Estas fases conforman un proceso iterativo, donde cada etapa colabora con la siguiente para minimizar el riesgo de errores, optimizar la utilización de recursos y asegurar que el producto final cumpla con los requerimientos planteados por el cliente o por el entorno en el que se despliegue.

En segundo lugar, exploraremos cómo herramientas de planificación como el pseudocódigo y los diagramas de flujo pueden aportar claridad al momento de estructurar la lógica de un algoritmo antes de comenzar la implementación. El pseudocódigo, al ser una representación textual simplificada, permite que el programador y el equipo de trabajo se centren en la secuencia de pasos y la resolución de la problemática sin verse distraídos por la sintaxis particular de un lenguaje de programación. Por su parte, los diagramas de flujo brindan una visión visual y global de los pasos que seguirá el software, facilitando la comunicación entre miembros del equipo y la detección temprana de inconsistencias o redundancias en la lógica. Esta combinación de recursos —pseudocódigo y diagramas— promueve un entendimiento compartido de la solución, lo cual es esencial para proyectos de ingeniería en ciberseguridad y gestión de TI, donde la colaboración y la precisión adquieren un papel protagónico.

2.1. **Fases del desarrollo de software: análisis, diseño, codificación, prueba.**

El proceso de desarrollo de software es un conjunto de etapas interrelacionadas que aseguran la creación de soluciones robustas y eficientes. Cada fase aporta actividades específicas que, al integrarse, permiten detectar y corregir errores de forma temprana, optimizar recursos y garantizar que el software cumpla con los requerimientos del usuario.

Para una mejor comprensión revise el video:

- Título del enlace relacionado: Aprende qué es Desarrollo de Software y sus etapas
- Descripción del enlace relacionado: Este video explora el fascinante proceso de transformar ideas en software funcional, detallando las etapas clave del análisis y desarrollo de software, desde la concepción de una idea hasta su implementación final.
- Enlace: Aprende qué es Desarrollo de Software y sus etapas (Clase fácil)

2.1.1. **Análisis**

La fase de análisis constituye el punto de partida en cualquier proyecto de desarrollo de software, ya que durante este período se definen con claridad el problema a resolver y los requerimientos específicos del usuario o cliente. A lo largo de esta etapa, el equipo responsable del proyecto recopila información mediante entrevistas, encuestas, observación directa y la revisión de documentos existentes, con el fin de entender a fondo las necesidades, expectativas y posibles restricciones. Este trabajo inicial es crucial porque sienta las bases de todo el proceso subsiguiente, evitando que el desarrollo se realice sobre suposiciones inexactas o metas poco definidas. Cuando el análisis se realiza de forma metódica, se establecen objetivos claros y se identifican los actores involucrados (por ejemplo, propietarios, empleados y clientes), lo que facilita la alineación de todos los participantes con la visión del proyecto.

Para ilustrarlo, en el caso de un sistema de gestión de inventario destinado a una tienda, el análisis abarcaría aspectos como la identificación de quienes usarán el sistema (propietario, personal de ventas, encargados de bodega), los requerimientos que permitirán controlar el stock de productos, registrar entradas y salidas de mercancía, y generar reportes de ventas. Además, en esta fase se suele llevar a cabo un estudio de viabilidad, tanto técnica como económica, que evalúa si las tecnologías disponibles, los plazos y el presupuesto son adecuados para desarrollar la solución propuesta. Si en este punto se descubren limitaciones insalvables (como

un presupuesto muy reducido o requerimientos imposibles de cumplir con la tecnología elegida), se pueden ajustar las expectativas antes de comprometer recursos en etapas más costosas.

Un análisis exhaustivo reduce la probabilidad de malentendidos o re-trabajos en fases posteriores, dado que, cuanto más claro sea el entendimiento del problema y de los objetivos, menos correcciones serán necesarias en el futuro. Por ello, la precisión y la profundidad en la recolección de datos son decisivas: cualquier error u omisión en la fase de análisis podría repercutir en todas las etapas siguientes (diseño, codificación y prueba), incrementando costos y retrasando la entrega del producto final. De esta manera, la etapa de análisis no solo delimita el alcance del proyecto, sino que también contribuye a que el diseño posterior se ajuste realmente a las necesidades detectadas, estableciendo una base sólida para un desarrollo exitoso.

Un análisis detallado y riguroso es la clave para un desarrollo de software exitoso.

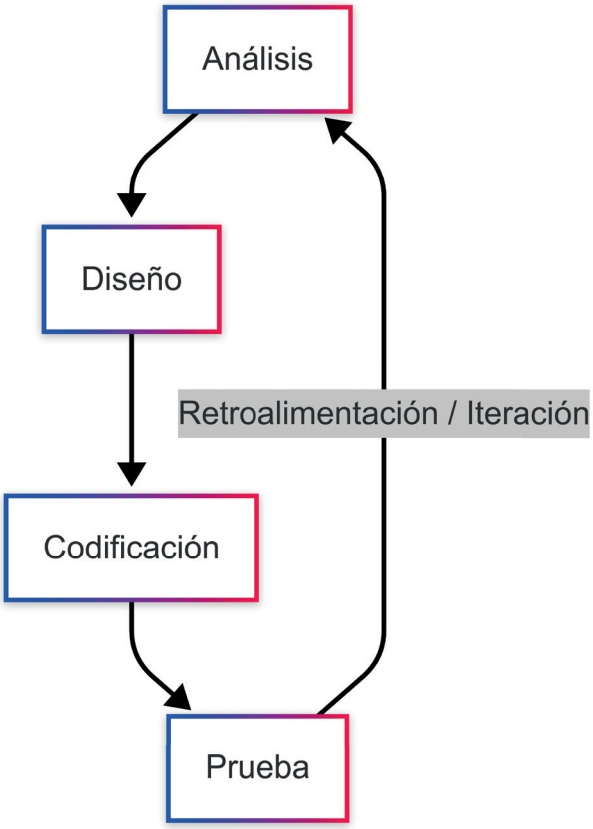


Imagen 1. *Diagrama Esquemático de las Fases del Desarrollo de Software*.

Damián Nicolalde Rodríguez. (2024). *Diagrama Esquemático de las Fases del Desarrollo de Software*.

2.1.2. **Diseño**

La fase de diseño es una etapa esencial en el desarrollo de software, donde se establece la estructura lógica y técnica que guiará el desarrollo del sistema previamente definido en la fase de análisis. En esta etapa se traducen los requerimientos y necesidades identificados durante el análisis en una solución técnica clara, organizada y eficiente. El diseño implica decisiones fundamentales sobre la arquitectura del software, la selección de algoritmos y la definición detallada de los componentes y relaciones que conformarán el sistema. Durante este proceso, se emplean diversas herramientas que permiten organizar y representar gráficamente la lógica del software, tales como diagramas de flujo, diagramas UML, esquemas de bases de datos y pseudocódigo.

Por ejemplo, si pensamos en el desarrollo de un sistema de inventario, el diseño implicaría inicialmente descomponer el sistema en módulos funcionales. Se podrían considerar módulos como: gestión de productos, control de entradas y salidas del almacén, módulo de reportes y módulo de usuarios. Luego, se elaboran diagramas de casos de uso que permiten visualizar claramente cómo interactúan los usuarios (por ejemplo, administradores y empleados de inventario) con cada uno de los módulos y qué acciones específicas pueden realizar. Esto facilita identificar las funciones clave y los procesos que el sistema debe soportar para cumplir sus objetivos.

Además, el diseño debe contemplar cómo estos módulos interactúan entre sí. Aquí es donde los diagramas de clases entran en juego. Estos diagramas describen los objetos o entidades que conforman el sistema (productos, categorías, ventas, empleados, etc.), mostrando claramente sus atributos y las relaciones existentes entre ellos, lo que ayuda a visualizar la organización lógica y las dependencias. Esto es crucial para determinar la estructura del código fuente en la siguiente fase (codificación), ya que establece claramente la relación y jerarquía entre diferentes componentes.

Otro aspecto fundamental en el diseño es la planificación del flujo lógico del programa mediante el uso del pseudocódigo. Este recurso permite describir detalladamente las operaciones del programa de manera clara y concisa, pero sin las limitaciones sintácticas propias de los lenguajes de programación. Por ejemplo, al implementar la lógica para registrar nuevos productos en un inventario, el pseudocódigo ayudaría a visualizar paso a paso cómo debería operar esta funcionalidad antes de escribir la codificación final, asegurando que la lógica sea robusta y sin ambigüedades.

Un buen diseño no solo optimiza el proceso de desarrollo, sino que también facilita la detección temprana de posibles errores, reduciendo significativamente los costos asociados a correcciones durante la fase de pruebas o incluso después del lanzamiento del sistema. Por tanto, dedicar tiempo suficiente y esfuerzo meticuloso al diseño es indispensable para asegurar la calidad y eficiencia del producto final.

Un diseño bien estructurado facilita la codificación y mejora la mantenibilidad del sistema.

Tabla 1: Cuadro Comparativo de las Fases del Desarrollo de Software

Fase	**Objetivo Principal**	**Ejemplo de Actividad**
Análisis	Definir el problema y recabar requerimientos.	Entrevistar usuarios, análisis de documentación.
Diseño	Planificar la solución y definir la arquitectura del sistema.	Elaboración de diagramas de casos de uso y de clases.
Codificación	Traducir el diseño a código fuente usando buenas prácticas.	Escribir módulos y funciones, pruebas unitarias.

Prueba	Verificar que el software cumple los requerimientos y funciona correctamente.	Pruebas unitarias, de integración y de sistema.

Damián Nicolalde (2024).

2.1.3. **Codificación**

La fase de codificación es un paso esencial en el desarrollo del software, donde las ideas plasmadas durante las etapas previas de análisis y diseño finalmente cobran vida a través de un lenguaje de programación específico. En esta fase, los desarrolladores escriben el código fuente siguiendo estrictamente los lineamientos definidos anteriormente, transformando las especificaciones teóricas en soluciones prácticas y operativas. Aquí, el énfasis recae en mantener el código claro, eficiente y bien documentado, asegurando que sea legible no solo para quien lo escribe, sino también para futuros desarrolladores que podrían mantener o ampliar el sistema.

Por ejemplo, retomando nuestro caso práctico del sistema de gestión de inventarios, durante la etapa de codificación se crean funciones individuales para cada aspecto esencial del sistema. Se desarrollan funciones específicas para registrar nuevos productos, actualizar cantidades en stock, y gestionar la salida de productos. También se escriben rutinas que permiten realizar consultas rápidas sobre el estado actual del inventario. Durante la codificación, se integran validaciones exhaustivas para prevenir errores comunes (como ingresar valores incorrectos) mediante estructuras condicionales (if-else) y manejo de excepciones (try-except), garantizando así la estabilidad del sistema en situaciones reales.

Además, en la fase de codificación, cobra importancia la aplicación de buenas prácticas de programación. Esto incluye la elección adecuada de nombres de variables utilizando convenciones como snake_case para mejorar la legibilidad y mantener consistencia en todo el código. La elección del lenguaje, en este caso Python, facilita enormemente el proceso gracias a su sintaxis clara, lo que permite que el código sea intuitivo y más sencillo de depurar y mantener.

Es común también que, durante esta fase, se implementen comentarios claros y concisos que expliquen la lógica del código, facilitando así la comprensión por parte de otros miembros del equipo o para futuras revisiones del software. Además, la incorporación de manejo de errores es clave. Esto incluye, por ejemplo, implementar bloques try-except que permitan gestionar adecuadamente excepciones o situaciones inesperadas como la introducción incorrecta de datos por parte del usuario.

Finalmente, aunque la fase de codificación parece ser una simple traducción del diseño, es en realidad una etapa crítica que requiere atención minuciosa a los detalles, ya que pequeños errores en esta etapa pueden propagarse y causar problemas significativos más adelante. Por lo tanto, es recomendable realizar revisiones constantes del código mediante técnicas como revisiones cruzadas entre compañeros (pair programming) o mediante herramientas automatizadas que faciliten la detección temprana de errores antes de avanzar a la fase de pruebas.

La calidad del código depende de la organización y la aplicación rigurosa de buenas prácticas de programación.

2.1.4. **Prueba**

La fase de pruebas es esencial en el desarrollo del software, ya que permite asegurar que el producto final cumple con los requerimientos establecidos en las etapas anteriores y funciona correctamente bajo distintos escenarios y condiciones. Es fundamental que el software se someta a un proceso exhaustivo de verificación antes de su implementación, identificando posibles fallos o inconsistencias que afecten la calidad del sistema. Durante esta etapa, se ejecutan diversas pruebas especializadas, cada una dirigida a un aspecto particular del software:

- Pruebas unitarias: Consisten en verificar cada componente del sistema de forma independiente, gen-

eralmente utilizando casos específicos y aislados. Permiten confirmar que cada módulo responde correctamente a entradas esperadas y maneja adecuadamente las situaciones extremas.

- Pruebas de integración: Evalúan el funcionamiento conjunto de varios módulos que ya fueron validados unitariamente. En estas pruebas, el objetivo es asegurarse de que la interacción entre los módulos no genere errores adicionales, ya sea por problemas de comunicación interna, interfaces incompatibles o lógicas conflictivas.
- Pruebas de sistema: Son pruebas más generales que se realizan al software completo, simulando un ambiente similar al entorno final donde será utilizado. Se enfoca en evaluar no solo las funcionalidades individuales, sino también aspectos como el rendimiento general, la usabilidad, la eficiencia y la estabilidad del sistema bajo distintas condiciones operativas.
- Pruebas de aceptación: Corresponden a la validación final antes de que el sistema sea entregado al usuario final. Se realizan junto con el usuario o cliente final, asegurando que la solución desarrollada cumple satisfactoriamente con las expectativas y requerimientos previamente establecidos.

Cada tipo de prueba juega un papel crucial para garantizar la estabilidad y fiabilidad del sistema final, ya que permite detectar errores en etapas tempranas cuando resulta menos costoso resolverlos. Las pruebas deben estar documentadas claramente, especificando los casos evaluados, resultados obtenidos, errores encontrados y soluciones implementadas. Además, se recomienda utilizar herramientas de automatización, como frameworks de testing (pytest en Python), para facilitar el proceso de ejecución y repetición sistemática de estas pruebas, especialmente en proyectos de gran escala. Esto asegura un seguimiento efectivo del estado del software y su evolución hacia un producto confiable y robusto.

La implementación adecuada y rigurosa de estas pruebas representa una inversión de tiempo significativa, pero crucial para reducir riesgos de fallos críticos en la operación diaria del software. En consecuencia, la fase de pruebas no solo valida la solución técnica, sino que también aumenta la confianza del usuario en la calidad y efectividad del producto final desarrollado.

Las pruebas rigurosas permiten identificar y corregir errores antes de que el software entre en producción.

Tabla 2: Características y Objetivos de Cada Fase del Desarrollo

Fase	**Características Clave**	**Objetivo Final**
Análisis	Recolección de datos, definición de requerimientos, delimitación del problema.	Establecer una base sólida para el proyecto.
Diseño	Creación de modelos, diagramas, planificación de la arquitectura.	Visualizar y estructurar la solución.
Codificación	Escritura de código, aplicación de buenas prácticas, modularización.	Traducir el diseño en un programa funcional.
Prueba	Ejecución de pruebas unitarias, integración, sistema y aceptación.	Garantizar la fiabilidad y eficiencia del software.

Damián Nicolalde (2024).

El proceso de desarrollo de software es iterativo y colaborativo. Cada fase – análisis, diseño, codificación y prueba – contribuye a la creación de soluciones sólidas, permitiendo detectar errores tempranamente y optimizar el uso de recursos. Una adecuada aplicación de estas fases asegura que el producto final cumpla con las expectativas del usuario y se mantenga a lo largo del tiempo.

2.2. **Pseudocódigo y diagramas de flujo.**

Las herramientas de planificación son esenciales para transformar ideas en soluciones de software. El

pseudocódigo y los diagramas de flujo permiten a los desarrolladores estructurar la lógica de un algoritmo de forma clara y organizada antes de proceder a la codificación.

2.2.1. **Pseudocódigo**

El pseudocódigo es una representación textual simplificada de un algoritmo. Se utiliza para describir la secuencia de pasos necesarios para resolver un problema sin la rigidez de la sintaxis de un lenguaje de programación. Su principal ventaja es que permite concentrarse en la lógica y la estructura del algoritmo.

Ejemplo de Pseudocódigo: Para calcular el área de un rectángulo.

```
INICIO
  LEER largo
  LEER ancho
  CALCULAR area = largo * ancho
  MOSTRAR area
FIN
```

Este ejemplo detalla de forma sencilla cada paso del proceso, haciendo hincapié en la lógica y en la estructura secuencial del algoritmo.

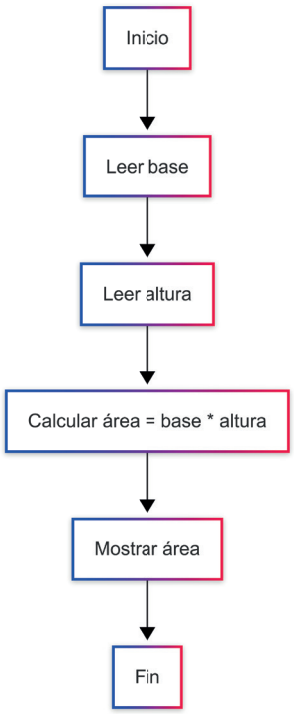


Imagen 2. Ejemplo Gráfico de Pseudocódigo para el Cálculo de Área.

Damián Nicolalde Rodríguez. (2024). Gráfico de Pseudocódigo para el Cálculo de Área.

2.2.2. **Diagramas de Flujo**

Los diagramas de flujo son representaciones gráficas que ilustran la secuencia de pasos y decisiones en un algoritmo. Utilizan símbolos estandarizados, tales como:

- Óvalos: para representar el inicio y el fin del proceso.
- Rectángulos: para procesos o acciones.
- Rombos: para decisiones (condiciones).
- Paralelogramos: para entradas y salidas.

Ejemplo: Para el problema del área de un rectángulo, el diagrama de flujo incluiría:

- Inicio (óvalo).
- Entrada de "largo" (paralelogramo).
- Entrada de "ancho" (paralelogramo).
- Proceso de cálculo (rectángulo).
- Salida del resultado (paralelogramo).
- Fin (óvalo).

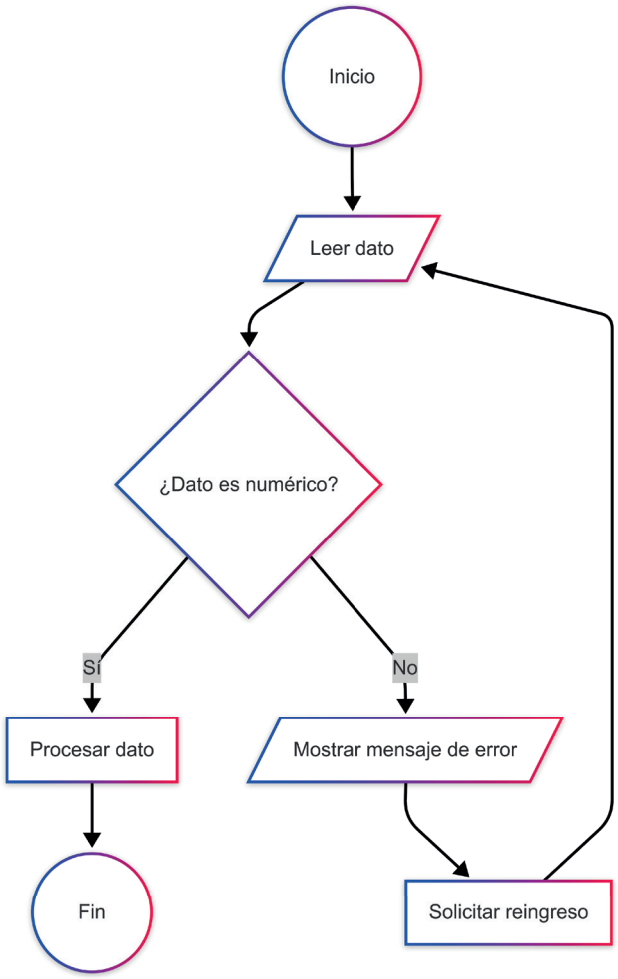


Imagen 3. Diagrama de Flujo para la Validación de Datos.

Damián Nicolalde Rodríguez. (2024). Diagrama de Flujo para la Validación de Datos.

2.2.3. **Integración de Pseudocódigo y Diagramas de Flujo**

El uso conjunto del pseudocódigo y los diagramas de flujo permite abordar la planificación de un algoritmo desde dos perspectivas complementarias: una textual y otra visual. Mientras el pseudocódigo ofrece un desglose detallado de cada paso, el diagrama de flujo permite visualizar la secuencia global de operaciones y decisiones. Esta integración facilita la identificación de errores y mejora la comunicación dentro del equipo.

Ejemplo de Integración: Si se desea crear un algoritmo para validar la entrada de datos numéricos, el pseudocódigo puede describir los pasos para leer, convertir y validar los datos, mientras que el diagrama de flujo mostrará gráficamente la bifurcación en función de si los datos son válidos o no.

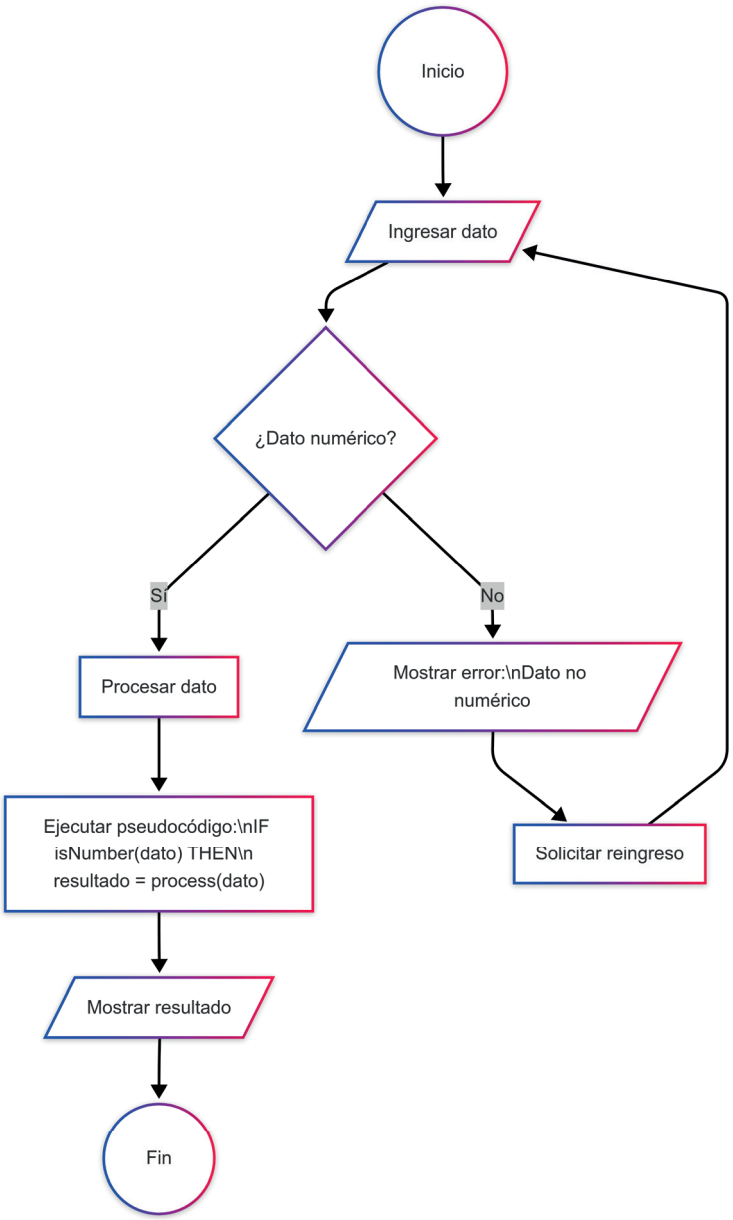


Imagen 4. Diagrama de Flujo Integrativo del Proceso de Desarrollo

Damián Nicolalde Rodríguez. (2024). Diagrama de Flujo Integrativo del Proceso de Desarrollo.

Para una mejor comprensión revise el video:

- Título del enlace relacionado: Programación: Diagramas de flujo y pseudocódigo
- Descripción del enlace relacionado: El video explica el diagrama de flujo como representación gráfica de procesos algorítmicos mediante símbolos estandarizados y su interacción con el pseudocódigo para diseñar lógica estructurada. Incluye un ejemplo práctico que integra ambos métodos, destacando su rol en la planificación de soluciones computacionales antes de la codificación formal.
- Enlace: Programación: Diagramas de flujo y pseudocódigo

La integración de las fases del desarrollo de software con el uso de herramientas de planificación es esencial para la creación de soluciones robustas y de alta calidad. Un análisis detallado y un diseño estructurado permiten que la codificación sea más ordenada y las pruebas sean efectivas, reduciendo la incidencia de errores y facilitando el mantenimiento futuro del software.

El pseudocódigo y los diagramas de flujo actúan como puentes entre la teoría y la práctica, permitiendo a los desarrolladores estructurar la lógica de un algoritmo sin las restricciones de la sintaxis de un lenguaje de programación. Mientras que el pseudocódigo se enfoca en detallar la secuencia de pasos de forma textual, el diagrama de flujo ofrece una representación visual que facilita la comunicación y el análisis colaborativo.

La implementación de estas herramientas en el proceso de desarrollo no solo mejora la calidad del software, sino que también optimiza el uso de recursos y fomenta la colaboración en equipo. En entornos complejos, como en la ingeniería en ciberseguridad y la gestión de TI, donde la precisión y la confiabilidad son críticas, estas metodologías se vuelven aún más relevantes.

El contenido presentado en esta clase, ofrece una base sólida para comprender el ciclo de vida del desarrollo de software y las herramientas que lo acompañan. La planificación y la estructuración meticulosa de los procesos son la clave para el éxito en la creación de software de calidad, y el dominio de estas técnicas es indispensable para cualquier profesional en el campo de la programación y la ciberseguridad.

Tabla 3: Resumen Comparativo – Pseudocódigo vs. Diagrama de Flujo

Aspecto	**Pseudocódigo**	**Diagrama de Flujo**
Formato	Texto, en lenguaje natural.	Gráfico, con símbolos estandarizados.
Propósito	Describir la lógica de forma secuencial y detallada.	Visualizar la secuencia de operaciones y decisiones.
Ventaja Principal	Facilidad para detallar la lógica sin errores de sintaxis.	Claridad en la representación visual del proceso.
Aplicación	Planificación y documentación de algoritmos.	Revisión, comunicación y análisis colaborativo de la lógica.

Damián Nicolalde (2024).

La importancia de planificar adecuadamente antes de comenzar la codificación. Un análisis riguroso y un diseño bien estructurado son la base de cualquier proyecto exitoso. Además, el uso de herramientas visuales fortalece la capacidad de identificar y corregir fallas en el proceso de desarrollo, lo que es especialmente crítico en entornos complejos como la ingeniería en ciberseguridad y la gestión de TI.

Referencias citadas en la Clase 2.

- https://elibro.puce.elogim.com/es/ereader/puce/230298
- https://puce.odilo.us/info/facil-aprendizaje-estructuras-de-datos-algoritmos-c-aprenda-facilmente-estructuras-de-datos-graficamente-03127232
- https://puce.odilo.us/info/aprende-c-en-un-fin-de-semana-03105596

Definición de los términos citados en la Clase 2.

Pseudocódigo: Es una representación simplificada y en lenguaje natural de un algoritmo o proceso. Permite describir la lógica y los pasos necesarios para resolver un problema sin requerir la precisión sintáctica de un lenguaje de programación formal. Se utiliza para planificar, analizar y comunicar la estructura de un algoritmo de manera clara y ordenada.

Diagrama de flujo: Es una representación gráfica que ilustra la secuencia de pasos, decisiones y procesos de un algoritmo. Utiliza símbolos estandarizados (como óvalos para el inicio y fin, rectángulos para procesos, rombos para decisiones y paralelogramos para entradas/salidas) para facilitar la visualización y comprensión de la lógica subyacente, ayudando a identificar errores y oportunidades de mejora en el flujo del proceso.

Profundización Clase 2.

Recurso_profundizacion_clase2.docx

PUCE Virtual

La excelencia no se improvisa

síguenos