UNIDAD III. CODIFICACIÓN, INTEGRACIÓN E IMPLANTACIÓN DE SISTEMAS

1.- CODIFICACIÓN.

Los estándares de codificación son un conjunto de recomendaciones que describen los procedimientos y la etiqueta ideales para crear código de computadora. Debido a la estabilidad que brindan estos estándares en todos los proyectos, es más sencillo para los desarrolladores comprender y mantener el código producido por otros. Además, seguir los estándares de codificación hace que el código sea inteligible, reutilizable y mantenible, lo que reduce los defectos y aumenta la calidad general.

1.1. Definición de estándar de Codificación.

- Estándares de Codificación en Ingeniería de Software: Un Enfoque Estructurado y "Ingeniería del Software" de Roger Pressman.

Roger Pressman considera que los estándares de codificación son una parte esencial de la ingeniería de software. Al adoptar y aplicar estándares de codificación adecuados, los desarrolladores pueden mejorar la calidad, la legibilidad y la mantenibilidad del software, lo que se traduce en un menor número de errores, un desarrollo más eficiente y un producto final más confiable.

Pressman resalta la importancia de mantener una consistencia en el estilo de codificación a lo largo de un proyecto. Esto facilita la comprensión del código por parte de todos los miembros del equipo de desarrollo y reduce la probabilidad de errores y aboga por escribir código que sea fácil de leer y comprender. Esto implica utilizar nombres de variables y funciones descriptivos, una correcta indentación del código, comentarios claros y concisos, y una estructura de código lógica y organizada.

- Estructura de datos en C++ de L Joyanes; L Sanchez; I Zahonero.

La codificación o implementación implica la traducción de la solución de diseño elegida en un programa de computadora escrito en un lenguaje de programación tal como C++ o Java. Si el análisis y las especificaciones han sido realizadas con corrección y los

algoritmos son eficientes, la etapa de codificación normalmente es un proceso mecánico y casi automático de buen uso de las reglas de sintaxis del lenguaje elegido.

- Systems Analysis and Design de Kendall & kendall.

La codificación consiste en convertir datos a un formato estandarizado que permita su fácil procesamiento y análisis por los sistemas de información. Este proceso también puede incluir la compresión de datos para optimizar el uso de recursos.

- Prácticas de Programación Extrema de Kent Beck y Martin Fowler.

Becky Fowler abordan la importancia de los estándares de codificación como una herramienta fundamental para la colaboración efectiva y la calidad del software en el contexto de la programación extrema (XP).

Los estándares de codificación son herramientas valiosas para los desarrolladores de software, ya que les permiten crear código de alta calidad que sea fácil de mantener, menos propenso a errores, eficiente en su ejecución y compatible con diferentes entornos de desarrollo. La adopción de estas normas en los proyectos de software conduce a un mejor desarrollo de software, lo que se traduce en productos más confiables, seguros y escalables.

Los estándares de codificación facilitan la comprensión del código entre los miembros de un equipo, permitiendo una mejor colaboración y evitando malentendidos. También ayudan a mantener un código consistente, legible y fácil de mantener, reduciendo la probabilidad de errores y mejorando la confiabilidad del software.

- Los estándares de codificacións según Epistic Technologies (pagina web: https://hackernoon.com).

1.2. Factores asociados con los estándares de codificación.

Los factores asociados a los estándares de codificación son diversos y abarcan aspectos técnicos, organizacionales y externos. Al considerar estos factores cuidadosamente, los equipos de desarrollo pueden elegir e implementar estándares de codificación que se adapten a las necesidades específicas de sus proyectos y mejoren la calidad general del software.

Los estándares de codificación son herramientas valiosas para los desarrolladores de software, ya que les permiten crear código de alta calidad que sea fácil de mantener, menos propenso a errores, eficiente en su ejecución y compatible con diferentes entornos de desarrollo. La adopción de estas normas en los proyectos de software conduce a un mejor desarrollo de software, lo que se traduce en productos más confiables, seguros y escalables.

Es importante recordar que no existe un único conjunto de estándares de codificación universalmente aceptado. Cada organización o proyecto puede elegir adoptar un conjunto de estándares que se adapte a sus necesidades específicas. Sin embargo, los principios y las mejores prácticas promovidas por los autores mencionados anteriormente sirven como una base sólida para la creación de estándares de codificación efectivos.

- Factores según Ian Sommerville.

Los factores asociados con los estándares de codificación incluyen la facilidad de mantenimiento, la reducción de errores, la eficiencia en la ejecución y la compatibilidad con diferentes entornos de desarrollo. Estos factores contribuyen a la creación de software de alta calidad. Factores Clave de los Estándares de Codificación Según Sommerville:

a. Facilidad de Mantenimiento: El código debe ser fácil de entender y modificar para corregir errores, actualizar funcionalidades y mejorar el rendimiento. Prácticas Comunes: Uso de comentarios explicativos, documentación completa, y un diseño modular.

b. Reducción de Errores: Minimizar los errores en el código desde el principio reduce los costos y el tiempo de desarrollo. Prácticas Comunes: Pruebas unitarias y de integración, revisiones de código, y adherencia a principios de programación segura.

c. Eficiencia en la Ejecución: Asegurar que el código se ejecute de manera eficiente es crucial para el rendimiento del sistema. Prácticas Comunes: Optimización del código, selección adecuada de estructuras de datos y algoritmos, y profiling para identificar cuellos de botella.

d. Compatibilidad: Garantizar que el software pueda interactuar con otros sistemas y plataformas es vital para su interoperabilidad. Prácticas Comunes: Uso de estándares abiertos, API bien definidas, y pruebas de compatibilidad en múltiples entornos.

e. Seguridad y Gestión de Riesgos: Proteger el software contra amenazas y vulnerabilidades es esencial para mantener la confianza del usuario y la integridad de los datos. Prácticas Comunes: Implementación de medidas de seguridad, auditorías de seguridad, y planes de gestión de riesgos.

- Código Limpio: Un manual de artesanía de software ágil de Robert C. Martin, donde identifica cuatro factores clave que influyen en la adopción e implementación exitosa de estándares de codificación:

a. Cultura de la empresa: La cultura de la empresa debe valorar la calidad del código como un elemento fundamental en el desarrollo de software. Esto implica reconocer la importancia de la legibilidad, mantenibilidad y confiabilidad del código, y considerarlas como objetivos prioritarios en el proceso de desarrollo.

b. Compromiso de la gerencia: La gerencia debe demostrar un liderazgo activo en la promoción de los estándares de codificación. Esto implica comunicar claramente la importancia de estos estándares, asignar recursos para su implementación y capacitación, y dar ejemplo a los desarrolladores siguiendo las pautas establecidas.

c. Capacitación y educación: La empresa debe ofrecer programas de capacitación formal sobre los estándares de codificación elegidos. Estos programas deben estar diseñados para proporcionar a los desarrolladores una comprensión profunda de las pautas establecidas y las mejores prácticas para su aplicación.

d. Herramientas y soporte: La empresa debe proporcionar herramientas de análisis estático de código que puedan identificar posibles violaciones de los estándares de codificación y sugerir mejoras en el estilo y la estructura del código.

Robert C. Martin enfatiza que la adopción exitosa de estándares de codificación requiere un esfuerzo conjunto que involucra a la cultura de la empresa, la gerencia, los desarrolladores y la organización en general. Al implementar las estrategias mencionadas anteriormente, las empresas pueden crear un entorno propicio para la mejora continua de la calidad del código y el desarrollo de software de alto nivel.

- Pressman destaca la importancia de los factores clave de los estándares de codificación para mejorar la calidad, la mantenibilidad y la legibilidad del software. Argumenta que los estándares de codificación:

a. Promueven la consistencia: Aseguran que el código se escriba de manera uniforme en todo el proyecto, facilitando su comprensión y modificación.

b. Mejoran la legibilidad: Hacen que el código sea más fácil de leer y entender para otros desarrolladores, incluso aquellos que no están familiarizados con el proyecto específico.

c. Reducen errores: Disminuyen la probabilidad de errores al establecer reglas claras para la estructura, la indentación, los nombres de variables y otras características del código.

d. Facilitan el mantenimiento: Simplifican el proceso de mantenimiento del código, ya que los desarrolladores pueden encontrar y modificar secciones específicas del código con mayor facilidad.

1.3. Ventajas de la aplicación de estándares de codificación

Los estándares de código son una serie de reglas definidas para un lenguaje de programación, o bien, un estilo de programación específico. El estilo garantiza que todos los ingenieros que contribuyen a un proyecto tengan una forma única de diseñar su código, lo que da como resultado una base de código coherente, asegurando una fácil lectura y mantenimiento.

El uso de estándares es muy importante en la calidad de software, sin embargo mantener todos los proyectos cumpliendo a la perfección con esto no es una tarea fácil, requiere un gran esfuerzo y constancia por parte del equipo de desarrollo. Mientras más y más compañías han adoptado estándares, todavía hay aquellas que realizan el desarrollo de sus proyectos sin ellos.

Seguir los estándares de codificación puede ayudar con el mantenimiento del software al hacer que el código sea más fácil de leer y depurar siguiendo convenciones comunes y mejores prácticas. Esto reduce el riesgo de introducir errores o incoherencias al aplicar la coherencia y la claridad en todo el código base.

Los estándares también facilitan la colaboración y la comunicación entre los desarrolladores al establecer un lenguaje y un estilo comunes para el código. Además, mejoran la calidad y la fiabilidad del código siguiendo principios y técnicas probadas para el diseño, las pruebas y la documentación. Por último, los estándares de codificación mejoran la reutilización y la portabilidad del código al adherirse a estándares que son ampliamente aceptados y respaldados por herramientas y plataformas.

Ventajas de la implementación de estándares de codificación

1. **Detección temprana de falla**s: Al buscar cumplir con los estándares que establezcas es más sencillo detectar posibles errores desde la revisión de código, evitando que esos problemas lleguen a producción.
2. **Reducción de la complejidad**: El cumplir con las reglas acerca del estilo de código ayuda a construir código más limpio, permitiendo detectar fácilmente oportunidades para simplificar funciones.
3. **Código de fácil lectura**: El respetar los estándares en un proyectos le permite a nuevos miembros del equipo acoplarse más fácilmente al ritmo de trabajo y a entender mejor el código de los proyectos existentes.
4. **Código reusable**: Contar con segmentos de código que pueden ser consumidos por más de un servicio, gracias al uso de buenas prácticas, hace menos frecuente la repetición de código.

1.4. Dificultades de la aplicación de estándares de codificación.

Los estándares de codificación no son una solución única para los problemas y desafíos de mantenimiento de software, y tienen sus propios inconvenientes y limitaciones. Por ejemplo, los estándares de codificación pueden reflejar las preferencias o sesgos personales de los creadores o adoptantes, en lugar de los méritos o necesidades objetivas del código o proyecto. También pueden ser rígidos y restrictivos, lo que dificulta la creatividad, la flexibilidad o la productividad de los desarrolladores. Además, los estándares de codificación pueden volverse obsoletos y obsoletos si no se mantienen al día con los cambios en la tecnología, la metodología o el dominio del proyecto de software. Por último, los desarrolladores o los usuarios no pueden hacer cumplir, supervisar o respetar los estándares de codificación, ya sea de forma intencionada o no.

Sommerville en su libro "Ingeniería del Software", identifica las siguientes dificultades para la aplicación de estándares de codificación:

1. **Resistencia al cambio:** Los desarrolladores pueden ser reacios a abandonar sus estilos de codificación establecidos y adoptar nuevos estándares. Esto puede deberse a la falta de comprensión de los beneficios de los estándares, la percepción de que son demasiado restrictivos o la simple comodidad con las prácticas actuales.
2. **Falta de familiaridad:** Los desarrolladores pueden no estar familiarizados con los estándares de codificación relevantes o no comprender cómo aplicarlos en su trabajo diario. Esto puede dificultar la implementación efectiva de los estándares.
3. **Costos de implementación:** La implementación de estándares de codificación puede requerir capacitación, herramientas y recursos adicionales. Estos costos pueden ser un obstáculo para algunas organizaciones.
4. **Incompatibilidad con sistemas existentes:** Los estándares de codificación pueden no ser compatibles con sistemas existentes, lo que requiere un esfuerzo adicional para adaptarlos o reescribirlos.
5. **Dificultad para medir el beneficio:** Puede ser difícil cuantificar el beneficio de aplicar estándares de codificación, lo que dificulta la justificación de su implementación ante los gerentes y las partes interesadas.
6. **Falta de soporte de la gerencia:** Si la gerencia no apoya la implementación de estándares de codificación, es probable que los esfuerzos de los desarrolladores sean en vano.
7. **Cambios en los estándares:** Los estándares de codificación pueden evolucionar con el tiempo, lo que requiere un esfuerzo continuo para actualizar las prácticas de codificación y la capacitación de los desarrolladores.

1.5. Partes importantes que se deben considerar en un estándar y codificación.

Codificación

A) Según Gary Bronson en su Libro C++ para Ingeniería y Ciencias, 2da Edición.

Este paso, el cual también se conoce como escribir el programa y poner en práctica la solución, consiste en traducir la solución de diseño elegida en un programa de computadora. Si los pasos de análisis y solución se han realizado en forma correcta, el paso de codificación se vuelve bastante mecánico. En un programa bien diseñado, los planteamientos que forman el programa se conformarán, sin embargo, con ciertos patrones o estructuras bien definidos en el paso de solución. Estas estructuras controlan la forma en que el programa se ejecuta y consiste en los siguientes tipos:

1. Secuencia

2. Selección

3. Iteración

4. Invocación

La secuencia define el orden en que son ejecutadas las instrucciones por el programa. La especificación de cuál instrucción entra primero, cuál en segundo lugar, etc., es esencial si el programa ha de lograr un propósito bien definido.

La selección proporciona la capacidad para hacer una elección entre diferentes operaciones, dependiendo del resultado de alguna condición. Por ejemplo, el valor de un número puede comprobarse antes que una división sea realizada. Si el número no es cero, puede usarse como el denominador de una operación de división; de lo contrario, la división no se ejecutará y se mostrará al usuario un mensaje de advertencia.

La iteración, la cual también se denomina bucle, ciclo o repetición, proporciona la capacidad para que la misma operación se repita con base en el valor de una condición. Por ejemplo, podrían introducirse y sumarse grados de manera repetida hasta que un grado negativo sea introducido. Ésta sería la condición que significa el fin de la entrada y adición repetitiva de grados. En ese punto podría ejecutarse el cálculo de un promedio para todos los grados introducidos.

La invocación implica invocar, o solicitar, un conjunto de instrucciones cuando sea necesario. Por ejemplo, el cálculo del pago neto de una persona implica las tareas de obtener las tarifas de salario y las horas trabajadas, calcular el pago neto y proporcionar un reporte o cheque por la cantidad requerida. Por lo general unas de estas tareas individuales se codificarían como unidades separadas que son llamadas a ejecución, o invocadas, según se necesiten.

B) Según Roger S. Pressman en su libro Ingeniería del Software: un enfoque práctico, 7ma Edición, pagina 94-95, dice que:

La actividad de construcción incluye un conjunto de tareas de codificación y pruebas que lleva a un software operativo listo para entregarse al cliente o usuario final. En el trabajo de ingeniería de software moderna, la codificación puede ser

1) la creación directa de lenguaje de programación en código fuente (por ejemplo, Java)

2) la generación automática de código fuente que usa una representación intermedia parecida al diseño del componente que se va a construir o

3) la generación automática de código ejecutable que utiliza un "lenguaje de programación de cuarta generación" (por ejemplo, Visual C++).

Principios de codificación. Los principios que guían el trabajo de codificación se relacionan de cerca con el estilo, lenguajes y métodos de programación. Sin embargo, puede enunciarse cierto número de principios fundamentales:

Principios de preparación: Antes de escribir una sola línea de código, asegúrese de:

- Entender el problema que se trata de resolver.
- Comprender los principios y conceptos básicos del diseño.
- Elegir un lenguaje de programación que satisfaga las necesidades del software que se va a elaborar y el ambiente en el que operará.
- Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo.
- Crear un conjunto de pruebas unitarias que se aplicarán una vez que se haya terminado el componente a codificar.

Principios de programación: Cuando comience a escribir código, asegúrese de:

- Restringir sus algoritmos por medio del uso de programación estructurada.
- Tomar en consideración el uso de programación por parejas.
- Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
- Entender la arquitectura del software y crear interfaces que son congruentes con ella.
- Mantener la lógica condicional tan sencilla como sea posible

- Crear lazos anidados en forma tal que se puedan probar con facilidad.
- Seleccionar nombres significativos para las variables y seguir otros estándares locales de codificación.
- Escribir código que se documente a sí mismo.
- Crear una imagen visual (por ejemplo, líneas con sangría y en blanco) que ayude a entender.

Principios de validación: Una vez que haya terminado su primer intento de codificación, asegúrese de:

- Realizar el recorrido del código cuando sea apropiado.
- Llevar a cabo pruebas unitarias y corregir los errores que se detecten.
- Rediseñar el código

Estándar

Los estándares ayudan a determinar si se están utilizando los métodos óptimos. La gestión adecuada del código puede facilitar su depuración y prueba. A medida que un proyecto avanza dentro de un equipo, la importancia de los estándares de codificación se vuelve cada vez más evidente. Con estándares en su lugar, el código es fácil de descifrar y sigue una estructura coherente.

La consistencia beneficia la integridad del programa, y debe mantenerse a lo largo del proceso de codificación. Además, es crucial asegurar que los estándares de codificación se adhieran de manera uniforme y no entren en conflicto en varios niveles del sistema. Finalmente, el código fuente del programa debe parecer como si hubiera sido producido por un solo programador en una sola sesión.

¿Por qué los estándares de codificación son importantes para la Calidad del Código?

Los estándares de codificación no solo sirven para mantener la consistencia. Las consecuencias de no mantener estos criterios pueden ser severas. Los desarrolladores podrían estar empleando sus propios enfoques si no se establecen medidas, lo que podría tener efectos negativos como:

- **Problemas de seguridad:** Si no se siguen estándares de calidad, el software podría ser vulnerable a amenazas externas o incluir fallos debido a un código deficiente.

- **Rendimiento:** La programación defectuosa puede afectar la calidad del producto final. El resultado es una mala experiencia de usuario y problemas evidentes con la seguridad y funcionalidad del sistema.

Mejores Prácticas de los Estándares de Codificación

Puede mejorar la calidad de su código siguiendo algunas de estas mejores prácticas en los estándares de codificación:

- **Legibilidad:** Dado que los códigos difíciles de leer requieren más tiempo y energía para que los desarrolladores los comprendan, es importante considerar la legibilidad. Para hacer el código más fácil de leer, trate de separar las diferentes partes del código dividiendo los bloques de código en párrafos. Usar la indentación para mostrar dónde comienzan y terminan las estructuras de control también es una buena idea. De esta manera, puede saber dónde está el código entre ellas.
- **Priorizar la Arquitectura:** Debido a la falta de tiempo, las personas tienden a apresurarse en las tareas y no tienen en cuenta la arquitectura. Sin embargo, en la gran mayoría de estas situaciones, las cosas salen mal. Es un esfuerzo a medio hacer escribir código sin considerar primero la arquitectura. Debe saber cómo funciona el código, qué hace, cómo puede usarse y cómo se prueba, depura y actualiza antes de comenzar.
- **Revisión del Código:** Muchas personas piensan que las revisiones de código son una excelente manera de enseñar a los nuevos miembros del equipo cómo trabajar en diferentes áreas del software. Sin embargo, frecuentemente pasan por alto el hecho de que el propósito de estas revisiones es mantener la calidad del código alta y no impartir conocimientos. Las revisiones de código han sido muy importantes para mantener la alta calidad del código y el crecimiento del equipo. Por lo tanto, es mejor asignar esta tarea a desarrolladores que comprendan al menos el 95% del código.

1.6. Herramientas utilizadas para la verificación de código.

De acuerdo a Ferrera. A, en su libro "métodos y usos aplicados en la auditoría de sistemas informáticos", primera edición, pág 456:

La revisión del código permite a una empresa asegurarse que los desarrolladores de aplicaciones sigan técnicas de desarrollo seguras. Una regla general es que una prueba de intrusión no debería descubrir ninguna vulnerabilidad adicional de la aplicación relacionada

con el código desarrollado después de que la aplicación se haya sometido a una revisión de código adecuada, o al menos, sólo deberían descubrirse vulnerabilidades de baja criticidad.

A la hora de efectuar este tipo de tareas se pueden diferenciar dos caminos. Por un lado, realizar las auditorías manuales, haciendo búsquedas de palabras clave, o leyendo el código fuente desde el principio hasta el final. Y la otra vertiente, es el uso de las herramientas automáticas.

Existen varias herramientas que se utilizan para la verificación de código, algunas de las más comunes son:

- **Linters:** Son herramientas que analizan el código en busca de posibles errores, problemas de estilo o prácticas no recomendadas. Algunos ejemplos de linters populares son ESLint para JavaScript, Pylint para Python y RuboCop para Ruby.
- **Compiladores y analizadores estáticos:** Revisan el código en busca de errores de sintaxis, problemas de tipo y otros posibles problemas. Algunos ejemplos son GCC para C/C++, TypeScript Compiler para TypeScript y Java Compiler para Java.
- **Pruebas unitarias:** Son una forma de verificar el comportamiento del código en pequeñas unidades aisladas. Se pueden utilizar frameworks como JUnit para Java, NUnit para .NET y Jest para JavaScript.
- **Revisión de código por pares:** La técnica implica que otro desarrollador revise el código en busca de posibles errores o problemas. Esto se puede hacer de forma manual o utilizando herramientas como GitHub's Pull Request feature o Gerrit.
- **Herramientas de análisis de cobertura:** Estas herramientas analizan qué partes del código están siendo ejecutadas durante las pruebas y proporcionan información sobre la cobertura del código. Ejemplos de herramientas de análisis de cobertura son JaCoCo para Java, Istanbul para JavaScript y Coverage.py para Python.

2.- INTEGRACIÓN.

En su libro "Ingeniería del Software: Un Enfoque Práctico", Roger Pressman (1995) define la integración como el proceso de combinar componentes de software en un sistema completo y probado. Este proceso implica ensamblar módulos individuales en una estructura coherente, verificando que funcionen juntos correctamente. La integración se lleva a cabo en diferentes niveles, desde la integración de unidades hasta la integración del sistema completo.

Pressman destaca que la integración es crucial para garantizar que el software funcione según lo previsto y cumpla con los requisitos establecidos. Durante este proceso, se

identifican y resuelven posibles conflictos entre los componentes, se realizan pruebas para validar el comportamiento del sistema en su conjunto y se asegura la interoperabilidad entre las partes. Una integración efectiva contribuye a la estabilidad, confiabilidad y calidad general del software desarrollado.

Además, Pressman enfatiza que la integración no es un evento único, sino un proceso continuo a lo largo del ciclo de vida del desarrollo de software. A medida que se agregan nuevas funcionalidades o se realizan modificaciones, es necesario volver a integrar los componentes existentes para mantener la cohesión y el rendimiento del sistema. La gestión adecuada de la integración es fundamental para lograr un producto final exitoso y satisfactorio para los usuarios finales.

2.1. Etapas para la integración de un sistema de información.

De acuerdo a Ferrera. A, en su libro "métodos y usos aplicados en la auditoría de sistemas informáticos", primera edición, pág 456:

Un proyecto de integración requiere una planificación amplia, diseño del sistema y desarrollo de software para garantizar una implementación exitosa. Las empresas pueden comenzar el proceso siguiendo siete pasos integrales:

- **Determinar los requisitos:** A lo largo de este proceso, las empresas trabajaran estrechamente con el departamento de TI y un desarrollador de software para garantizar que se cumplan sus necesidades de integración. Pero primero, un equipo de gestión de la empresa debe reunir una lista de requisitos que su sistema integrado debe abordar. Este esquema debe mencionar cualquier operación, proceso de negocio o conocimiento que la empresa desee mejorar. Con esta información, el desarrollador puede construir un sistema fácil de usar con los componentes necesarios para mejorar el rendimiento empresarial.
- **Análisis de conducta:** Una vez completada la lista de requisitos, el departamento de TI la analizará para determinar la viabilidad del diseño de software que aborde adecuadamente todas las preocupaciones. En esta fase, el negocio debe estar preparado para hacer concesiones, ya que pueden sugerir otras funciones que creen que la empresa necesitará eventualmente. Aunque inicialmente puede costar más agregar características, si la compañía escatima en el diseño de software y luego necesita una mejor funcionalidad, debe invertir en un sistema completamente nuevo. Por lo tanto, es más rentable prepararse demasiado para las necesidades potenciales de la empresa.

- **Diseñar infraestructura de software:** Una vez que el análisis se lleva a cabo y se aprueba, el desarrollador de software comienza a construir la arquitectura del sistema. Los planos detallados diseñan el diseño de cómo los distintos sistemas integrarán y agregan datos.
- **Desarrollar un plan de gestión:** Una vez diseñada la infraestructura, el equipo de administración debe colaborar con las demás partes para calcular los riesgos, crear una línea de tiempo y determinar opciones alternativas. Esta preparación asegurara que las operaciones puedan continuar funcionando sin problemas incluso si los desarrolladores tienen problemas o requieren una extensión.
- **Diseño Integración del Sistema:** El diseño de la integración del sistema es la fase más laboriosa y que consume mucho tiempo, ya que describe el proceso, las pruebas del sistema, los métodos y la logística. El éxito de este paso depende en gran medida de lo bien que se realizaron los pasos anteriores y si se cubrieron todos los detalles necesarios. Con una integración adecuada, todos los sistemas se integraran sin problemas sin perder ningún dato durante las transferencias.
- **Implementar la solución:** Después de diseñar y probar la solución de software para garantizar la calidad, está lista para ser implementada. Una vez completado el proceso de integración, la dirección puede comenzar a capacitar a los empleados en el nuevo sistema.
- **Realizar comprobaciones de mantenimiento:** Tal vez el paso más pasado por alto es realizar el mantenimiento rutinario en los integradores de sistemas. Los usuarios deben ejecutar diagnósticos programados en su software para asegurarse de que no se han cometido nuevos errores y que las funciones mejoran su rendimiento. Si surge algún problema, se puede contactar con el equipo de TI para solucionar problemas técnicos o agregar más componentes.

2.2. Diagramas utilizados para la Integración del Software.

De acuerdo al sitio web, AnyConnector, tipos de diagramas integrados, hay que considerar:

Existen varios tipos de diagramas que se utilizan para la integración del software, algunos de los más comunes son:

- **Diagrama de Arquitectura de Software:** Este tipo de diagrama muestra la estructura general del sistema de software, incluyendo los componentes principales, sus

interacciones y las interfaces entre ellos. Es útil para visualizar cómo se integran los diferentes módulos y componentes del sistema.

- **Diagrama de Secuencia:** Este tipo de diagrama muestra la secuencia de interacciones entre los diferentes componentes del sistema en un escenario específico. Es útil para visualizar cómo se comunican los diferentes elementos durante la ejecución del sistema.
- **Diagrama de Flujo de Datos:** Este tipo de diagrama muestra cómo se mueven los datos a través del sistema, incluyendo cómo se transforman y procesan en cada etapa. Es útil para comprender cómo se integran los diferentes procesos y funciones del sistema.
- **Diagrama de Despliegue:** Este tipo de diagrama muestra la disposición física de los componentes del sistema en el entorno de implementación, incluyendo servidores, dispositivos y redes. Es útil para visualizar cómo se distribuyen y comunican los diferentes elementos del sistema.
- **Diagrama de Componentes:** Este tipo de diagrama muestra los componentes del sistema y las dependencias entre ellos. Es útil para visualizar cómo se organizan y relacionan los diferentes elementos del sistema.

2.3. Pruebas de Integración.

Las pruebas de integración son una técnica sistemática para construir la arquitectura del software mientras se llevan a cabo pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño.

Con frecuencia existe una tendencia a intentar la **integración no incremental,** es decir, a construir el programa usando un enfoque de big bang. Todos los componentes se combinan por adelantado. Todo el programa se prueba como un todo. La corrección se dificulta pues el aislamiento de las causas se complica por la vasta extensión de todo el programa. Una vez corregidos estos errores, otros nuevos aparecen y el proceso continúa en un bucle aparentemente interminable.

La integración incremental es la antítesis del enfoque big bang. El programa se construye y prueba en pequeños incrementos, donde los errores son más fáciles de aislar y corregir; las interfaces tienen más posibilidades de probarse por completo; y puede aplicarse un enfoque de prueba sistemático. A continuación se presentaran algunas estrategias de integración incremental:

A) **Integración descendente.** La prueba de integración descendente es un enfoque incremental a la construcción de la arquitectura de software. Los módulos se integran al moverse hacia abajo a través de la jerarquía de control, comenzando con el módulo de control principal (programa principal). Los módulos subordinados al módulo de control principal se incorporan en la estructura en una forma de primero en profundidad o primero en anchura. Pasos para hacer integración descendente:

1) El módulo de control principal se usa como un controlador de prueba y los representantes (stubs) se sustituyen con todos los componentes directamente subordinados al módulo de control principal.
2) Dependiendo del enfoque de integración seleccionado (es decir, primero en profundidad o anchura), los representantes subordinados se sustituyen uno a la vez con componentes reales.
3) Las pruebas se llevan a cabo conforme se integra cada componente.
4) Al completar cada conjunto de pruebas, otro representante se sustituye con el componente real.
5) Las pruebas de regresión (que se analizan más adelante en esta sección) pueden realizarse para asegurar que no se introdujeron nuevos errores.

El proceso continúa desde el paso 2 hasta que se construye toda la estructura del programa.

- La estrategia de integración descendente verifica los principales puntos de control o de decisión al principio en el proceso de prueba. **En una estructura de programa "bien factorizada"**, la toma de decisiones ocurre en niveles superiores en la jerarquía y, por tanto, se encuentra primero.
- Pareciera que la estrategia descendente no tiene complicaciones, pero, en la práctica, pueden surgir **problemas** logísticos. El más común de éstos ocurre cuando se requiere procesamiento en niveles bajos en la jerarquía a fin de probar de manera adecuada los niveles superiores. Los representantes (stubs) sustituyen los módulos de bajo nivel al comienzo de la prueba descendente; por tanto, ningún dato significativo puede fluir hacia arriba en la estructura del programa. A la persona que realiza la prueba le quedan tres opciones: 1) demorar muchas pruebas hasta que los representantes se sustituyan con módulos reales, 2) desarrollar resguardos que realicen funciones limitadas que simulan al módulo real o 3) integrar el software desde el fondo de la jerarquía y hacia arriba.

B) **Integración ascendente.** La prueba de integración ascendente, como su nombre implica, comienza la construcción y la prueba con módulos atómicos (es decir, componentes en los niveles inferiores dentro de la estructura del programa). Puesto que los componentes se integran de abajo hacia arriba, la funcionalidad que proporcionan los componentes subordinados en determinado nivel siempre está disponible y se elimina la necesidad de representantes (stubs). Una estrategia de integración ascendente puede implementarse con los siguientes pasos:

1) Los componentes en el nivel inferior se combinan en grupos (en ocasiones llamados construcciones o builds) que realizan una subfunción de software específica.
2) Se escribe un controlador (un programa de control para pruebas) a fin de coordinar la entrada y salida de casos de prueba.
3) Se prueba el grupo.
4) Los controladores se remueven y los grupos se combinan moviéndolos hacia arriba en la estructura del programa.

C) **Pruebas de regresión.** Cada vez que se agrega un nuevo módulo como parte de las pruebas de integración, el software cambia. Se establecen nuevas rutas de flujo de datos, ocurren nuevas operaciones de entrada/salida y se invoca nueva lógica de control. Dichos cambios pueden causar problemas con las funciones que anteriormente trabajaban sin fallas. En el contexto de una estrategia de prueba de integración, la prueba de regresión es la nueva ejecución de algún subconjunto de pruebas que ya se realizaron a fin de asegurar que los cambios no propagaron efectos colaterales no deseados.

- Las pruebas de regresión ayudan a garantizar que los cambios (debidos a pruebas o por otras razones) no introducen comportamiento no planeado o errores adicionales.
- Conforme avanza la prueba de integración, el número de pruebas de regresión puede volverse muy grande. Por tanto, la suite de pruebas de regresión (el subconjunto de pruebas que se va a ejecutar) debe diseñarse para incluir solamente aquellas que aborden una o más clases de errores en cada una de las funciones del programa principal.

D) **Pruebas de humo.** La prueba de humo es un enfoque de prueba de integración que se usa cuando se desarrolla software de producto. Se diseña como un mecanismo de

ritmo para proyectos críticos en el tiempo, lo que permite al equipo del software valorar el proyecto de manera frecuente. En esencia, el enfoque de prueba de humo abarca las siguientes actividades:

1) Los componentes de software traducidos en código se integran en una construcción. Una construcción incluye todos los archivos de datos, bibliotecas, módulos reutilizables y componentes sometidos a ingeniería que se requieren para implementar una o más funciones del producto.
2) Se diseñan una serie de pruebas para exponer los errores que evitarán a la construcción realizar adecuadamente su función. La intención debe ser descubrir errores "paralizantes" que tengan la mayor probabilidad de retrasar el proyecto.
3) La construcción se integra con otras construcciones, y todo el producto (en su forma actual) se somete a prueba de humo diariamente. El enfoque de integración puede ser descendente o ascendente.

- La prueba de humo debe ejercitar todo el sistema de extremo a extremo. No tiene que ser exhaustiva, pero debe poder exponer los problemas principales. La prueba de humo debe ser suficientemente profunda para que, si la construcción pasa, pueda suponer que es suficientemente estable para probarse con mayor profundidad.

La selección de una estrategia de integración depende de las características del software y, en ocasiones, del calendario del proyecto. En general, un enfoque combinado (a veces llamado prueba sándwich), que usa pruebas descendentes para niveles superiores de la estructura del programa acopladas con pruebas ascendentes para niveles subordinados, puede ser el mejor arreglo.

Conforme se realiza la integración, quien efectúa la prueba debe identificar **los módulos críticos.** Un módulo crítico tiene una o más de las siguientes características: 1) aborda muchos requerimientos de software, 2) tiene un alto nivel de control (reside relativamente alto en la estructura del programa), 3) es complejo o proclive al error o 4) tiene requerimientos de rendimiento definidos. Los módulos críticos deben probarse tan pronto como sea posible. Además, las pruebas de regresión deben enfocarse en la función del módulo crítico.

Productos de trabajo de las pruebas de integración. Un plan global para integración del software y una descripción de las pruebas específicas se documentan en una Especificación

de pruebas. Este producto de trabajo incorpora un plan de prueba y un procedimiento de prueba, y se vuelve parte de la configuración del software. La prueba se divide en fases y construcciones que abordan características del software funcionales y de comportamiento específicas.

3.- IMPLANTACIÓN DE SISTEMAS.

La Implantación de sistemas es un tema relevante en lo que se refiere al desarrollo de software y de tecnologías de la información. A pesar de ello, la ingeniería de software continúa centrándose en abordar los problemas del desarrollo desde la mejora de procesos, pero sin abordar de manera sistemática la Implantación como un conjunto de temas específicos a ser tratados. Uno de los problemas detectados en gran parte de los proyectos informáticos, en general está dado por las dificultades en la Implantación de los mismos en los diferentes entornos sociales y tecnológicos, siendo esta etapa, un atributo fundamental para el éxito de la puesta en marcha de los sistemas. La investigación que se expone en el presente artículo se propone enmarcar los límites de la Implantación de sistemas, como parte un proceso inherente a la definición del proceso software, y que, asimismo, debe ser definido específicamente y a través de un conjunto de principios básicos que permitan comprender y abordar esta etapa como un área específica dentro de la Ingeniería de Software o la Ingeniería de Sistemas.

La correcta Implantación de un sistema involucra diversos aspectos tecnológicos, así como de contexto social en cuanto a los actores involucrados en las diferentes instancias. Para los proyectos de TI existen diversos modelos de proceso y de gestión que dividen en subprocesos cada una de las actividades que deben llevarse a cabo en el desarrollo y la puesta en marcha de los sistemas de información.

Es la ultima fase del desarrollo de sistemas. Consiste en instalar equipos o software nuevo, como resultado de un análisis y diseño previo como resultado de la sustitución o mejoramiento de la forma de llevar a cabo un proceso automatizado.

Al Implantar un sistema de información lo primero que debemos hacer es asegurarnos que el sistema sea operacional o sea que funcione de acuerdo a los requerimientos del análisis y permitir que los usuarios puedan operarlo.

Existen varios enfoques de Implementación:

- Es darles responsabilidad a los grupos.
- Uso de diferentes estrategias para el entrenamiento de los usuarios.

- El analista de sistemas necesita ponderar la situación y proponer un plan de conversión que sea adecuado para la organización.
- El analista necesita formular medidas de desempeño con las cuales evaluar a los usuarios.
- Debe convertir físicamente el sistema de información antigüo, al nuevo modificado.

En la preparación de la Implantación, aunque el sistema este bien diseñado y desarrollado correctamente su éxito dependerá de su implantación y ejecución por lo que es importante capacitar al usuario con respecto a su uso y mantenimiento.

Según ChatGPT:

La implantación de sistemas en los sistemas de información es un proceso complejo y multifacético que abarca diversas etapas y actividades destinadas a integrar nuevas tecnologías, aplicaciones, o sistemas dentro de una organización. Este proceso es crítico para asegurar que la inversión en tecnología sea eficaz y que los sistemas se utilicen de manera óptima. A continuación, se desglosan las principales etapas y consideraciones en la implantación de sistemas de información:

Etapas de la Implantación de Sistemas de Información

1. Planificación:
   - Análisis de Necesidades: Identificar los requerimientos del negocio y determinar los objetivos específicos que el nuevo sistema debe cumplir.
   - Evaluación de Opciones: Examinar diferentes soluciones tecnológicas, comparando sus funcionalidades, costos, y beneficios.
   - Planificación del Proyecto: Definir el alcance del proyecto, elaborar cronogramas, asignar recursos, y establecer un presupuesto.
2. Desarrollo y Configuración:
   - Diseño del Sistema: Crear especificaciones detalladas y arquitecturas del sistema que se van a implementar.
   - Personalización y Configuración: Ajustar el software o el sistema para que se alinee con los procesos y necesidades específicas de la organización.
   - Desarrollo de Software: En caso de que el sistema sea desarrollado internamente, esta etapa incluye codificación, pruebas unitarias, e integración de componentes.
3. Pruebas:
   - Pruebas de Sistema: Evaluar el sistema completo para asegurarse de que todas las partes funcionan correctamente.

• Pruebas de Usuario: Involucrar a usuarios finales para validar que el sistema cumple con sus necesidades y expectativas.

• Corrección de Errores: Identificar y resolver problemas antes del despliegue completo del sistema.

4. Implementación:

• Migración de Datos: Transferir datos de sistemas antiguos al nuevo sistema, asegurando la integridad y consistencia de los datos.

• Despliegue del Sistema: Implementar el sistema en el entorno de producción, lo cual puede hacerse de manera gradual o en un único paso, dependiendo de la estrategia adoptada.

• Capacitación: Entrenar a los usuarios finales y administradores del sistema para garantizar una transición suave y eficaz.

5. Post-Implementación y Mantenimiento:

• Soporte Técnico: Proveer soporte continuo a los usuarios para resolver problemas y responder preguntas.

• Mantenimiento y Actualizaciones: Realizar ajustes y mejoras continuas al sistema para mantener su relevancia y eficacia.

• Evaluación Post-Implementación: Revisar el proyecto para identificar lecciones aprendidas y asegurar que se han alcanzado los objetivos iniciales.

Consideraciones Importantes

• Gestión del Cambio: La introducción de nuevos sistemas puede encontrar resistencia por parte de los empleados. Es vital gestionar el cambio de manera efectiva, comunicando los beneficios del nuevo sistema y proporcionando apoyo durante la transición.

• Seguridad de la Información: Asegurar que el nuevo sistema cumpla con los estándares de seguridad para proteger la información sensible y cumplir con las normativas legales.

• Escalabilidad y Flexibilidad: El sistema debe ser capaz de crecer y adaptarse a las necesidades futuras de la organización.

• Integración con Sistemas Existentes: Asegurarse de que el nuevo sistema puede integrarse adecuadamente con los sistemas y aplicaciones existentes para evitar silos de información.

La implantación de sistemas en los sistemas de información es una tarea que requiere una planificación cuidadosa, ejecución precisa y una atención continua para asegurar el éxito a largo plazo y la máxima rentabilidad de la inversión tecnológica.

3.1. Adiestramiento de usuarios.

El adiestramiento de usuarios es el proceso de capacitar a las personas para que utilicen de manera efectiva y eficiente un sistema, software, o herramienta tecnológica específica. Este tipo de formación es esencial en organizaciones que implementan nuevos sistemas de información o actualizan los existentes, asegurando que los empleados puedan aprovechar al máximo las funcionalidades y beneficios de estas tecnologías.

Objetivos del Adiestramiento de Usuarios

1. **Competencia Técnica:** Proveer el conocimiento y las habilidades necesarias para operar el sistema de manera competente.

2. **Maximización del Uso**: Asegurar que los usuarios conozcan y utilicen todas las funcionalidades del sistema para optimizar su trabajo.

3. **Reducción de Errores**: Minimizar errores y malentendidos en el uso del sistema, lo que puede mejorar la eficiencia y la calidad del trabajo.

4. **Aumento de la Productividad**: Facilitar que los usuarios realicen sus tareas de manera más rápida y eficiente.

5. **Adaptación al Cambio**: Facilitar la transición y reducir la resistencia al cambio cuando se introduce un nuevo sistema o se actualiza uno existente.

Componentes del Adiestramiento de Usuarios

1. **Evaluación de Necesidades:** Identificar qué conocimientos y habilidades necesitan los usuarios para utilizar el sistema.

2. **Diseño de Programas de Formación**: Crear planes de formación que incluyan objetivos claros, contenidos relevantes y métodos de enseñanza adecuados.

3. **Entrega de la Formación:** Implementar el programa de formación a través de diversos métodos, como clases presenciales, e-learning, webinars, y manuales.

4. **Evaluación y Retroalimentación:** Medir la efectividad del adiestramiento mediante evaluaciones y recoger retroalimentación de los usuarios para mejorar continuamente el programa de formación.

5. **Soporte Continuo**: Proveer asistencia y recursos adicionales para los usuarios después del adiestramiento inicial, como sistemas de ayuda en línea y mesas de ayuda.

Métodos Comunes de Adiestramiento

- **Cursos Presenciales:** Clases impartidas por instructores donde los usuarios pueden interactuar y resolver dudas en tiempo real.

- **E-learning:** Cursos en línea que permiten a los usuarios aprender a su propio ritmo.

- **Webinars:** Sesiones en línea en vivo que ofrecen formación interactiva.

- **Manuales y Tutoriales:** Documentación y vídeos que los usuarios pueden consultar cuando lo necesiten.

- **Sistemas de Ayuda Integrados:** Herramientas de ayuda y tutoriales dentro del propio sistema.

Beneficios del Adiestramiento de Usuarios

- **Mayor Eficiencia:** Los usuarios pueden realizar sus tareas más rápido y con mayor precisión.

- **Mejora de la Moral:** Los empleados se sienten más competentes y seguros en su trabajo.

- **Reducción de Costos:** Menos errores y mayor eficiencia pueden reducir los costos operativos.

- **Mejora en la Adopción de Tecnología:** Una mejor comprensión y habilidad en el uso de nuevos sistemas facilita su adopción.

Desafíos del Adiestramiento de Usuarios

- **Resistencia al Cambio:** Los usuarios pueden resistirse a aprender nuevos sistemas.

- **Diversos Niveles de Habilidad:** Los usuarios pueden tener diferentes niveles de competencia técnica, lo que requiere programas de formación adaptados.

- **Actualización Continua:** Los sistemas de información cambian y se actualizan, lo que requiere un adiestramiento continuo.

El adiestramiento de usuarios es un componente crítico para el éxito de la implementación de sistemas de información en cualquier organización. Proporciona a los empleados las herramientas necesarias para utilizar las tecnologías de manera eficiente, mejorando así la productividad y la calidad del trabajo.

3.2. Prueba del sistema por parte de los usuarios.

Las pruebas de aceptación de usuarios, pruebas UAT o User Acceptance Testing, conocidas por las siglas en inglés, UAT, suponen un paso esencial en el desarrollo de software, por dos razones: porque son la última prueba antes del lanzamiento; y, como tal, involucra directamente a los usuarios finales de dicho software. Las pruebas de aceptación de usuario son una forma crucial de verificar si un producto o servicio cumple con los requisitos específicos de los clientes o usuarios finales.

Cada software se construye en base a requisitos o necesidades específicas. Por ello, el propósito de una prueba UAT es garantizar que se cumpla el requisito. También es útil para el equipo de pruebas porque el usuario o cliente puede probar el software y proporcionar comentarios para mejorarlo.

Así garantizará que el producto no solo sea de alta calidad sino también relevante para los requisitos del usuario.Este tipo de prueba se realiza al final del desarrollo del proyecto, cuando se ha completado la construcción y se está listo para su lanzamiento. El objetivo es asegurarse de que el producto funcione de manera efectiva y cumpla con los requisitos del usuario, lo que a su vez aumenta la confianza en el producto antes de su lanzamiento al mercado. Las pruebas de aceptación de usuario se realizan normalmente por los usuarios finales o clientes, ya que ellos son los que conocen mejor sus requisitos y necesidades. Sin embargo, también pueden ser realizadas por un equipo de pruebas independiente o incluso por un equipo de aceptación de usuario interno en la empresa.

3.3. Aprobación de los resultados de la prueba.

1. Aprobación de los resultados de prueba

La prueba del sistema es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo el sistema basado en computadora. Aunque cada prueba tenga un propósito diferente, todo él funciona para verificar que los elementos del sistema se hayan integrado de manera adecuada y que se realicen las funciones asignadas. Se estudian los tipos de pruebas del sistema que valen la pena para los sistemas basados en software.

Prueba de recuperación

La recuperación es una prueba del sistema que fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada. Si la recuperación es automática (realizada por el sistema en sí), se evalúa el reinicio, los mecanismos de puntos de verificación, la recuperación de datos y la reanudación para correcciones. Si la recuperación requiere intervención humana, se evalúa el tiempo medio de reparación (TMR) para determinar si está dentro de límites aceptables.

Pruebas de seguridad

La prueba de seguridad intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia. Para citar a Beizar [Bei84]: "La seguridad del sistema debe, desde luego, probarse para ser invulnerable ante ataques frontales; pero también debe probarse su invulnerabilidad contra ataques laterales y traseros."

Durante la prueba de seguridad, quien realiza la prueba juega el papel del individuo que desea penetrar al sistema. ¡Cualquier cosa vale! Quien realice la prueba puede intentar adquirir contraseñas por medios administrativos externos; puede atacar el sistema con software a la medida diseñado para romper cualquier defensa que se haya construido; puede abrumar al sistema, y por tanto negar el servicio a los demás; puede causar a propósito errores del sistema con la esperanza de penetrar durante la recuperación; puede navegar a través de datos inseguros para encontrar la llave de la entrada al sistema.

Pruebas de esfuerzo

La prueba de esfuerzo ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales. Por ejemplo, pueden 1) diseñarse pruebas especiales que generen diez interrupciones por segundo, cuando una o dos es la tasa promedio, (2) aumentarse las tasas de entrada de datos en un orden de magnitud para determinar cómo responderán las funciones de entrada, 3) ejecutarse casos de prueba que requieran memoria máxima y otros recursos, 4) diseñarse casos de prueba que puedan causar thrashing (que es un quebranto del sistema por hiperpaginación) en un sistema operativo virtual, 5) crearse casos de prueba que puedan causar búsqueda excesiva por datos residentes en disco. En esencia, la persona que realiza la prueba intenta romper el programa.

Una variación de la prueba es una técnica llamada prueba de sensibilidad.

Pruebas de rendimiento

La prueba de rendimiento se diseña para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado. La prueba del rendimiento ocurre a lo largo de todos los pasos del proceso de prueba. Incluso en el nivel de unidad, puede accederse al rendimiento de un módulo individual conforme se realizan las pruebas. Sin embargo, no es sino hasta que todos los elementos del sistema están plenamente integrados cuando puede determinarse el verdadero rendimiento de un sistema.
Las pruebas de rendimiento con frecuencia se aparean con las pruebas de esfuerzo y por lo general requieren instrumentación de hardware y de software, es decir, con frecuencia es necesario medir la utilización de los recursos (por ejemplo, ciclos del procesador) en forma meticulosa.

Pruebas de despliegue

En muchos casos, el software debe ejecutarse en varias plataformas y bajo más de un entorno de sistema operativo. La prueba de despliegue, en ocasiones llamada prueba de configuración, ejercita el software en cada entorno en el que debe operar, el software de instalación especializado (por ejemplo, “instaladores”) que usarán los clientes, así como toda la documentación que se usará para introducir el software a los usuarios finales.

Roger Pressman

La aprobación de los resultados de prueba en la implantación de sistemas es un paso crucial para garantizar que el sistema funcione correctamente y cumpla con los requisitos establecidos. Este proceso implica revisar detenidamente los resultados de las pruebas realizadas para asegurarse de que el sistema se comporta según lo esperado y que no hay errores críticos que puedan afectar su funcionamiento.

Una vez que los resultados de las pruebas han sido aprobados, se procede a la implementación del sistema en el entorno de producción.

La aprobación de los resultados de prueba es un proceso que requiere la participación de diferentes partes interesadas, como los responsables de negocio, los usuarios finales y los equipos de desarrollo y pruebas. Es fundamental que todos estén alineados y satisfechos con los resultados antes de proceder con la implantación del sistema

3.4. Liberación del sistema

Internet

1. Liberación del sistema

Es un proceso de poner en funcionamiento el sistema en el entorno de producción después de que se hayan completado con éxito las pruebas y se haya obtenido la aprobación de los resultados. La liberación implica mover el sistema desde el entorno de pruebas o desarrollo al entorno de producción, donde estará disponible para su uso por parte de los usuarios finales.

Durante el proceso de liberación del sistema, se deben seguir una serie de pasos y procedimientos para garantizar que la transición se realice de manera segura y efectiva. Estos pasos pueden incluir la preparación del entorno de producción, la instalación y configuración del sistema, la migración de datos si es necesario, la realización de pruebas adicionales en el entorno de producción y la capacitación de los usuarios finales.

Pasos:

Una aplicación Web está compuesta por un conjunto de archivos, páginas, módulos y código ejecutable, que se invocan o ejecutan dentro del ámbito de un directorio virtual (y sus subdirectorios) en el servidor Web de aplicaciones IIS. Para poner en marcha el sistema debemos seguir las políticas del II, las cuales incluyen, la coordinación y apoyo con el área de servidores Windows, que como se detalló en el capítulo es la encargada de administrar servidores de bases de datos y aplicaciones desarrolladas para y por el II, pasos que se describen a continuación.

- **Base de datos**

  Para cargar la base de datos “CMP_SISOL”, se creó un script con la información necesaria para generar la base de datos que incluye tablas, vistas, procedimientos almacenados, funciones, etc.; así como creación de usuarios y sus

correspondientes permisos. Para ejemplificar este paso, la imagen siguiente muestra el script ejecutado, en el entorno SQL Server 2008 del servidor de pruebas.

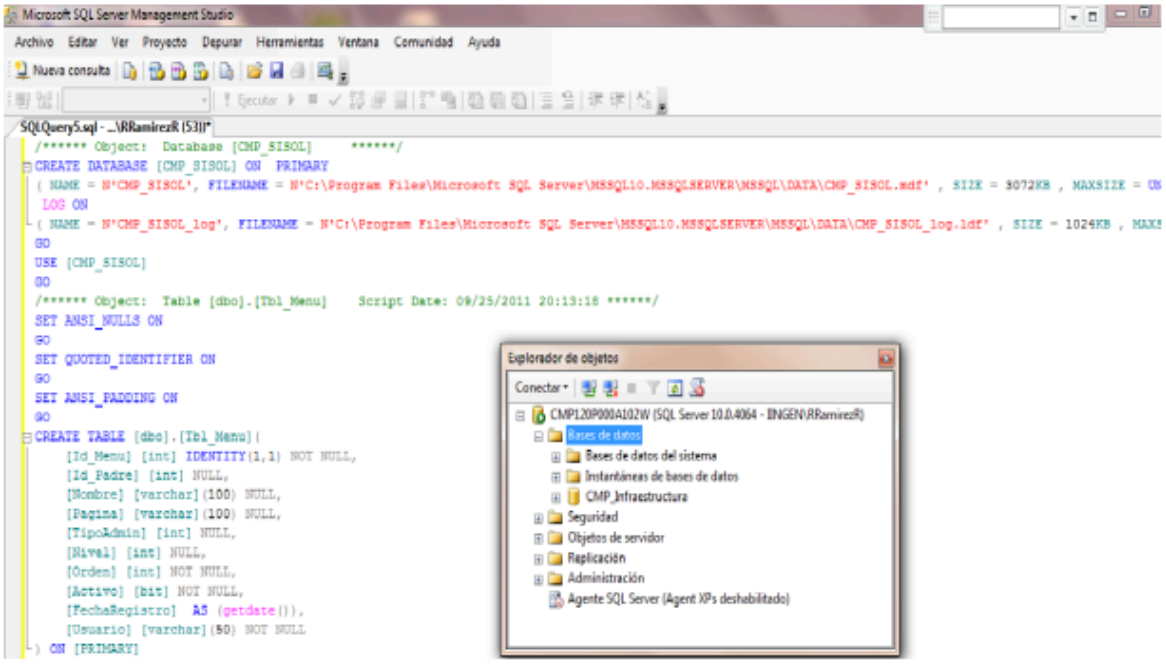

Figura 6.5 Script para la creación de la base de datos

- **Probar conexión**

  La cuenta de usuarios necesaria para el acceso a la base de datos es: II_CMP_SISOL, la cual es creada y especificada en el script del paso 1. Para probar la conexión debemos conectarnos al motor de base de datos indicando tipo de autenticación de SQL Server, cuenta y contraseña de usuario II_CMP_SISOL para inicio de sesión. Ver figura 6.6 para ejemplificar este punto.

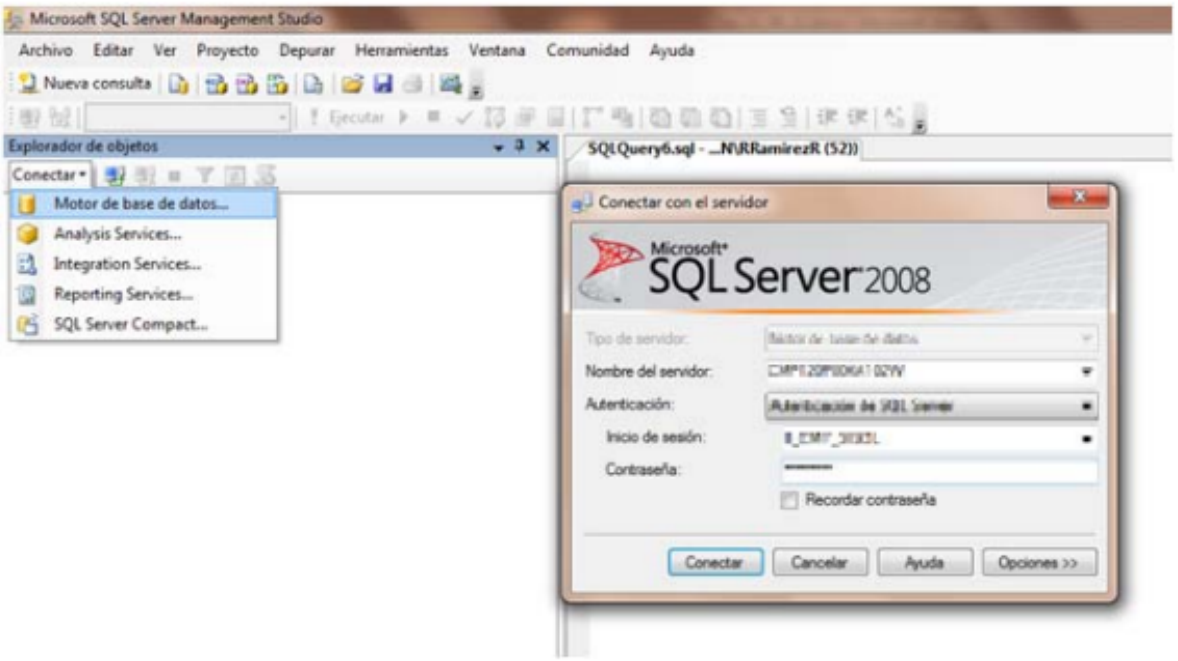

Figura 6.6 Acceso y autenticación a la base de datos

La conexión exitosa se confirma con una pantalla similar a la figura 6.7.

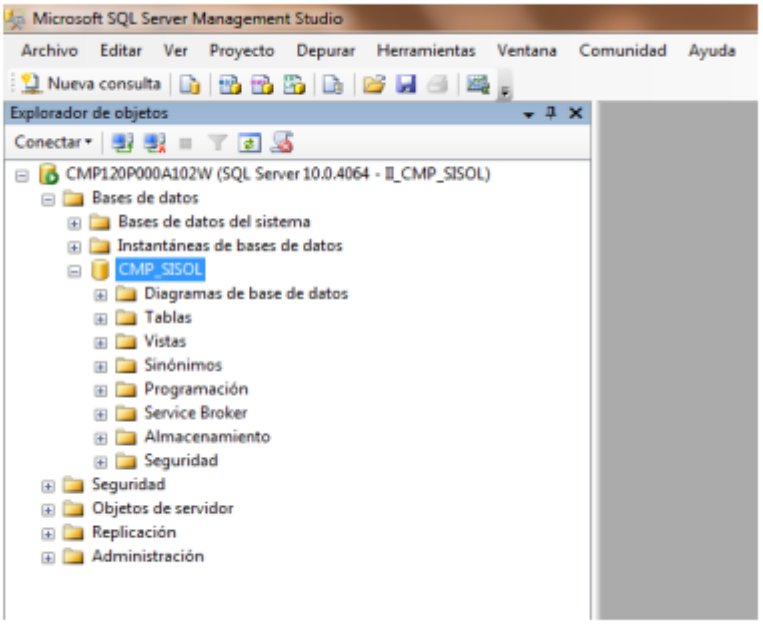

Figura 6.7 Prueba de conexión a la base de datos

- **Organización de aplicación en servidor de producción**

La organización de carpetas y permisos para nuestra aplicación, se detalla en el documento de especificaciones requerido por el área de Windows para su adecuada creación y administración del sitio virtual y grupos de control de usuarios. Dicho

documento se puede consultar en el anexo B.

- **Mantenimiento**

  Una vez realizados los puntos anteriores, el sistema está liberado y es necesario establecer un plan de mantenimiento de acuerdo con las políticas del II y los integrantes de la CSC responsables de su administración. Según la terminología ANSI-IEEE, el mantenimiento del software es: "la modificación de un producto software después de su entrega al cliente o usuario para corregir defectos, para mejorar el rendimiento u otras propiedades deseables, o para adaptarlo a un cambio de entorno".

4. Investigación:

4.1 ¿ Qué es Código Fuente?

El código fuente es el conjunto de instrucciones y declaraciones escritas por un programador utilizando un lenguaje de programación específico. Estas instrucciones son comprensibles para los seres humanos y están diseñadas para ser entendidas y procesadas por una computadora, aunque no directamente.

El código fuente incluye todas las directivas, funciones, algoritmos y estructuras de control que definen cómo debe comportarse un programa o sistema de software. Es la base desde la cual se compila un programa para crear un archivo ejecutable, que es la forma en que el software se distribuye y se ejecuta en las computadoras.

Aquí algunos puntos clave sobre el código fuente:

1. Lenguajes de Programación: El código fuente se escribe en lenguajes de programación como Python, Java, C++, JavaScript, entre muchos otros. Cada lenguaje tiene su propia sintaxis y reglas.

2. Legibilidad: Aunque los lenguajes de programación están diseñados para ser lógicos y comprensibles por humanos, también están estructurados de manera que las computadoras puedan interpretarlos y ejecutarlos una vez que se han traducido a lenguaje máquina (a través de un proceso llamado compilación o interpretación).

3. Control de Versiones: En el desarrollo de software, es común usar sistemas de control de versiones (como Git) para gestionar y rastrear los cambios en el código fuente. Esto permite a los desarrolladores colaborar de manera eficiente y mantener un historial de modificaciones.

4. Licencias y Derechos: El código fuente puede estar protegido por derechos de autor y puede estar sujeto a diversas licencias que determinan cómo se puede usar, modificar y distribuir. Ejemplos de estas licencias incluyen licencias de software libre y de código abierto como la GPL (General Public License) o la MIT License.

5. Importancia en el Desarrollo de Software: El código fuente es esencial para el desarrollo y mantenimiento de cualquier software. Es la representación más directa de las funcionalidades y lógica del programa.

4.2 ¿Qué es un Programa Fuente?

Un "programa fuente" es esencialmente un conjunto completo de código fuente que, cuando se compila o se interpreta, se convierte en un programa ejecutable o una aplicación funcional. A diferencia del código fuente individual que puede consistir en fragmentos o módulos aislados, un programa fuente incluye todo el código necesario para que el software realice las tareas y funciones para las cuales fue diseñado.

Aquí algunos puntos clave sobre un programa fuente:

1. Componentes del Programa Fuente:

- Archivos de Código: Múltiples archivos de código fuente escritos en uno o varios lenguajes de programación.
- Bibliotecas y Dependencias: Puede incluir referencias a bibliotecas externas o internas que proporcionan funcionalidades adicionales.
- Archivos de Configuración: Archivos que configuran cómo debe comportarse el programa, como archivos de configuración, variables de entorno, etc.
- Documentación: Instrucciones y descripciones que explican cómo funciona el código y cómo se debe utilizar o mantener.
- Scripts de Compilación: Instrucciones sobre cómo compilar el código fuente en un programa ejecutable.

2. Proceso de Desarrollo:

- Escritura: Los desarrolladores escriben el código fuente utilizando un entorno de desarrollo integrado (IDE) o un editor de texto.
- Pruebas: El programa fuente se prueba para asegurar que cumple con los requisitos y no contiene errores.
- Compilación/Interpretación: El código fuente se convierte en código máquina a través de la compilación (para lenguajes compilados como C++) o se ejecuta directamente a través de un intérprete (para lenguajes interpretados como Python).

- Depuración: Se identifican y corrigen errores en el código fuente.

3. Mantenimiento y Evolución:

- Actualizaciones: Los programas fuente se actualizan regularmente para mejorar funcionalidades, corregir errores y adaptar el software a nuevas necesidades.
- Control de Versiones: Uso de sistemas como Git para gestionar cambios y colaboraciones.

4. Distribución:

- Ejecutable: El resultado final del programa fuente, después de la compilación, es el ejecutable que los usuarios pueden instalar y ejecutar.
- Código Fuente Abierto: Algunos programas se distribuyen con su código fuente disponible para que otros desarrolladores puedan estudiar, modificar y mejorar el software (p.ej., software de código abierto).

4.3 Tipos de Lenguaje

Lenguaje Compilado

- Definición: Un lenguaje compilado es aquel cuyo código fuente se traduce completamente a un código máquina ejecutable por la computadora antes de que se ejecute. Este proceso se realiza mediante un compilador.

- Ejemplos: C, C++, Rust, Fortran.

- Características:

- Rendimiento: Suelen tener un rendimiento muy alto, ya que el código se convierte directamente en instrucciones de máquina.
- Verificación: Los errores de sintaxis y algunos errores lógicos se detectan en el momento de la compilación.
- Distribución: El resultado de la compilación es un archivo binario que se puede ejecutar sin necesidad de que el usuario tenga el compilador instalado.

Lenguaje Intermedio

Definición: Un lenguaje intermedio es un paso intermedio entre el código fuente y el código máquina. El código fuente se compila a un lenguaje intermedio (generalmente independiente de la plataforma), y luego este código intermedio se interpreta o se compila en tiempo de ejecución.

Ejemplos: Java (con bytecode), C# (con MSIL - Microsoft Intermediate Language).

Características:

- Portabilidad: El código intermedio es independiente de la plataforma, lo que permite que el mismo código se ejecute en diferentes sistemas operativos mediante una máquina virtual (por ejemplo, JVM para Java).
- Optimización: Permite ciertas optimizaciones que pueden realizarse tanto en la fase de compilación como en la de ejecución.
- Ejecución: El código intermedio se convierte en código máquina en tiempo de ejecución, generalmente a través de una máquina virtual o un compilador JIT (Just-In-Time).

Lenguaje Interpretado

Definición: Un lenguaje interpretado es aquel cuyo código fuente se ejecuta directamente por un intérprete, sin necesidad de una compilación previa a código máquina.

Ejemplos: Python, JavaScript, Ruby, PHP.

Características:

- Facilidad de Desarrollo: Los programas se pueden probar y modificar rápidamente, ya que no se requiere una fase de compilación.
- Portabilidad: El mismo código fuente puede ejecutarse en cualquier plataforma que tenga el intérprete adecuado.
- Rendimiento: Generalmente, el rendimiento es menor en comparación con los lenguajes compilados, ya que cada línea de código se analiza y ejecuta en tiempo real.

Diferencias y Funcionamiento de los Lenguajes Intermedios

- Diferencias:

- **Lenguaje Compilado:** Se traduce completamente a código máquina antes de la ejecución.
- **Lenguaje Intermedio:** Se compila a un código intermedio y luego se ejecuta mediante una máquina virtual.
- **Lenguaje Interpretado:** Se ejecuta línea por línea directamente del código fuente por un intérprete.

-Funcionamiento de los Lenguajes Intermedios:

1. **Compilación a Código Intermedio:** El código fuente se traduce a un lenguaje intermedio (como bytecode en Java o MSIL en C#).
2. **Ejecución en Máquina Virtual:** Este código intermedio se ejecuta en una máquina virtual (JVM para Java, CLR para .NET).

3. Compilación Just-In-Time (JIT): Opcionalmente, el código intermedio se puede convertir a código máquina justo antes de la ejecución para mejorar el rendimiento.

4.4 Interprete y Compilador

Intérprete

- Definición: Un intérprete es un programa que ejecuta instrucciones de código fuente directamente, traduciéndolas a medida que las ejecuta. No produce un archivo ejecutable separado.

- Funcionamiento:

- Lee el código fuente.
- Traduce cada instrucción a código máquina en tiempo real.
- Ejecuta cada instrucción inmediatamente después de traducirla.

Compilador

- Definición: Un compilador es un programa que traduce el código fuente de un lenguaje de programación a código máquina o a un lenguaje intermedio, creando un archivo ejecutable o un archivo de código intermedio.

- Funcionamiento:
- Analiza el código fuente completo.
- Traduce el código fuente a código máquina o a un lenguaje intermedio.
- Genera un archivo ejecutable o un archivo de código intermedio que se puede ejecutar posteriormente.