ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Fundamentos de ingeniería de software

Un enfoque práctico

Jenny A. Ruiz R. y Xavier I. Lucio M.

Fundamentos de ingeniería de software - Un enfoque práctico

Jenny Alexandra Ruiz Robalino y Xavier Iván Lucio Moreno

Primera edición electrónica: junio, 2025
ISBN: 978-9942-652-26-3

Revisión científica:
Dr. Raúl Hernán Rosero Miranda - Escuela Superior Politécnica de Chimborazo
Ph.D. Omar Gómez - Escuela Superior Politécnica de Chimborazo

Universidad de las Fuerzas Armadas-ESPE
Crnl. de C.S.M. Víctor Villavicencio A., Ph. D.
Rector

Publicación autorizada por:
Comisión Editorial de la Universidad de las Fuerzas Armadas-ESPE
Cpfg. Joseph Alexander Guamán Seis, Ph.D.
Presidente

Corrección de estilo y maquetación
Mtr. Xavier Chinga

Imagen de cubierta
https://lc.cx/H_BYhE

Universidad de las Fuerzas Armadas-ESPE
Av. General Rumiñahui s/n, Sangolquí, Ecuador
www.espe.edu.ec

Departamento de Ciencias de la Computación

Fundamentos de Ingeniería de Software

Un enfoque práctico

Jenny A. Ruiz R. y Xavier I. Lucio M.

EDITORIAL

UNIVERSIDAD DE LAS FUERZAS ARMADAS - ESPE

Jenny Alexandra Ruiz Robalino
jaruiz@espe.edu.ec

Docente a tiempo completo del Departamento de Ciencias de la Computación de la Universidad de Fuerzas Armadas – ESPE, Sangolquí-Ecuador.

Se graduó como Ingeniera de Sistemas e Informática en la Universidad Técnica de Ambato (UTA). Luego, obtuvo su maestría en Ciencias de Informática en la misma universidad; una segunda maestría en Ingeniería de Software en la Escuela Politécnica del Ejército.

Como docente en más de veinte años ha impartido varias cátedras en la universidad y desde el año 2020 ha iniciado con una línea de investigación enfocada en mejorar el proceso de enseñanza aprendizaje de la determinación de requisitos funcionales mediante el uso de herramientas básicas de la calidad como las 5W + 2H. Ha sido ponente en congresos nacionales y ha publicado artículos técnicos científicos en revistas nacionales e internacionales.

Xavier Iván Lucio Moreno
xavier.lucio@hotmail.es

Profesional independiente que colabora con varias empresas en áreas relacionadas a la administración de activos, mejora continua, y estandarización y eficiencia energética.

Se graduó como Ingeniero en Administración de Procesos en la Escuela Politécnica Nacional (EPN). Luego, obtuvo su maestría en Gestión de la Calidad y la Productividad en la Escuela Politécnica del Ejercito (ESPE).

Durante su ejercicio profesional ha implementado herramientas de mejora continua, GMAO, TPM, Trabajo Estandarizado, ISO 50001, y herramientas de Manufactura Lean. Se ha desempeñado también como instructor en temas relacionados con la administración de la calidad y como implementador de herramientas de la calidad.

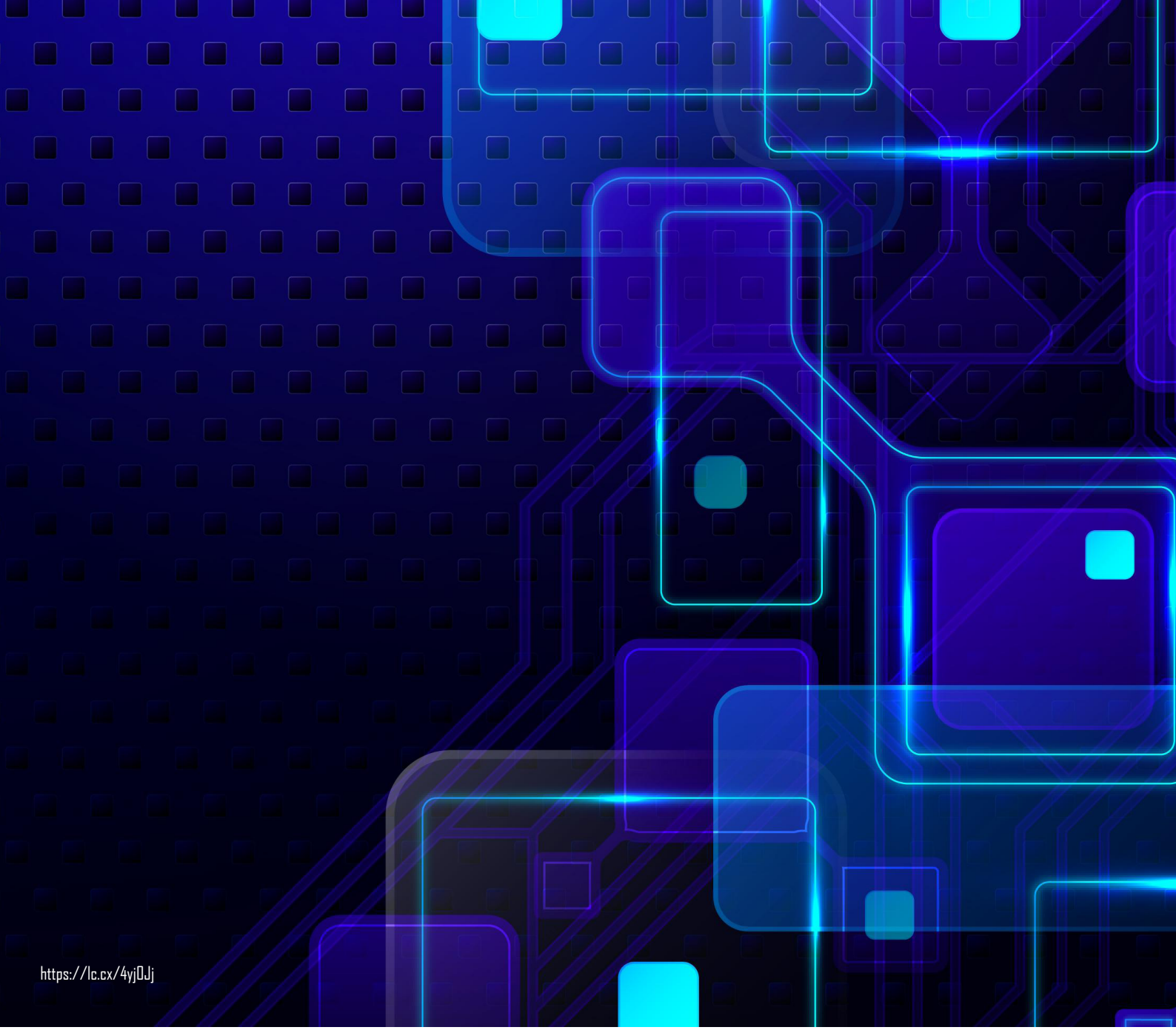
Agradezco de manera especial a la Universidad de las Fuerzas Armadas - ESPE y al Departamento de Ciencias de la Computación, sus autoridades, docentes y sobre todo a los estudiantes de todas las modalidades por permitirme ejercer mi vocación de docente.

A mi esposo y a mi hija, que son mi más grande motivación...

Índice

Índice de tablas

Índice de figuras



CAPÍTULO I

Fundamentos de la Ingeniería de Software

Perspectiva histórica

Para entender el término ingeniería del software conviene revisar lo que varios autores proponen acerca de su origen. Más de uno indica que este término aparece formalmente en la Conferencia de la OTAN de 1968 sobre ingeniería del software y fue referenciado por Friedrich Bauer. Otros, sin embargo, han señalado a la carta de 1966 de Anthony Oettinger, contenida en el documento de Comunicaciones de la ACM, como la primera aparición de este término. En esta se utilizó "ingeniería de software" para hacer una distinción sustancial entre la informática y la construcción de sistemas intensivos en software (Hinojosa, 2019).

La primera persona a quien se le atribuye el uso del término por primera vez es Margaret Hamilton, quien después de haber trabajado en el programa SAGE (Semi-automatic Ground Environment), se convirtió en la desarrolladora principal de Skylab y Apollo, mientras trabajaba en el "Draper Lab" (Piattini, 2016).

Abordando el tema cronológicamente, la historia inicia en los años treinta y cuarenta, tiempo en el que el software es un complemento del hardware; se profundizan, entonces, los trabajos de Alan Turing, Konrad Zuse y otros pioneros de la computación (Acevedo, 2014).

Para los años cincuenta, se aplica al desarrollo de software el mismo proceso de desarrollo de hardware, es decir, sobre la base de la experiencia de quienes lo realizan. El software en esta década es considerado como un producto añadido y la programación de computadores es todo un arte, para el cual no existen métodos sistemáticos. El desarrollo de software se realizaba sin ninguna planificación, una sola persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba (Hinojosa, 2019).

El diseño en sí mismo era un proceso implícito que se realizaba en la mente del programador y en el cual la documentación simplemente no existía (Piattini, 2016).

En la década de los sesenta, con el surgimiento de la multiprogramación y de los sistemas multiusuario, se implantan nuevos conceptos de interacción hombre-máquina, los sistemas en tiempo real recogían, analizaban y transformaban datos de múltiples fuentes para apoyar la toma de decisiones y, como consecuencia, nace la primera generación de sistemas de gestión de bases de datos (Ledesma, 2018).

Esta era se caracteriza por la aparición del software como producto y por el nacimiento de la fábrica de software, lugar en donde se producían programas con miles de líneas de código fuente que tenían que ser creadas, escritas, revisadas, corregidas cuando se detectaban fallas y modificadas cuando cambiaban los requisitos. En esta época, se fomenta el proceso de desarrollo de software tipo que codifica y corrige (Piattini, 2016).

A lo largo de las décadas de los setenta y ochenta, se crea una gran variedad de nuevas técnicas y métodos de ingeniería de software, tales como, la programación estructurada y el desarrollo orientado a objetos (Piattini, 2016).

En la década de los noventa y comienzos del nuevo siglo, la concurrencia (paralelismo y distribución) adquiere mayor relevancia, la orientación a objetos se extiende a las fases de análisis y diseño, dando origen al primer lenguaje de modelamiento unificado (UML) y se propone, además, el primer proceso de desarrollo orientado a objetos conocido como proceso unificado de desarrollo (RUP).

En la primera década del siglo veintiuno se firma el "Manifiesto ágil" como intento para reducir la complejidad de las metodologías existentes y, en respuesta a los modelos "pesados" tipo CMM (Modelo de madurez de software), surgen los métodos híbridos que buscan un equilibrio combinando la adaptabilidad de los ágiles con la formalidad y la documentación de los métodos rigurosos. Actualmente, vivimos el auge de este tipo de métodos, especialmente de Scrum; ha sido necesario modernizar a los ingenieros de software en la "cultura" y en las técnicas ágiles (Piattini, 2016).

Para la década del 2010, además de afianzarse las líneas de acción logradas en años anteriores, asistimos a una mayor integración entre la ingeniería del software y la ingeniería de sistemas, destacando el papel de los requisitos no funcionales y sobre todo de la seguridad (Piattini, 2016).

El futuro de la ingeniería de software se presenta muy retador. Para comprenderlo se puede revisar un breve resumen de algunas tendencias y retos abordados en la Conferencia "The Future of Software Engineering" (Oktaba, 2018).

Definiciones

El significado que tiene la ingeniería de software se divide en dos palabras: ingeniería y software. Un ingeniero está preparado para construir un producto software de alta calidad bajo restricciones de tiempo y presupuesto. El ingeniero se enfrenta a menudo con problemas mal definidos y con soluciones parciales, y tiene que apoyarse en métodos empíricos para evaluar soluciones. En la literatura se encuentra una larga historia de proyectos de software que fallaron (Figura 1). Ante esta realidad, la ingeniería de software se enfoca precisamente en la solución de problemas propios de la naturaleza cambiante y compleja del software (Charette, 2005).

Figura 1

Errores de software por su mala calidad

YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.
1999	United Way	Administrative processing system canceled after $12 million is spent.
1999	State of Mississippi	Tax system canceled after $11.2 million is spent; state receives $185 million damages.
1999	Hershey Foods Corp.	Problems with ERP system contribute to $151 million loss.
1998	Snap-on Inc.	Problems with order-entry system contribute to revenue loss of $50 million.
1997	U.S. Internal Revenue Service	Tax modernization effort canceled after $4 billion is spent.
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.
1996	Arianespace [France]	Software specification and design errors cause $350 million Ariane 5 rocket to explode.
1996	FoxMeyer Drug Co.	$40 million ERP system abandoned after deployment, forcing company into bankruptcy.
1995	Toronto Stock Exchange [Canada]	Electronic trading system canceled after $25.5 million** is spent.
1994	U.S. Federal Aviation Administration	Advanced Automation System canceled after $2.6 billion is spent.
1994	State of California	DMV system canceled after $44 million is spent.
1994	Chemical Bank	Software error causes a total of $15 million to be deducted from 100 000 customer accounts.
1993	London Stock Exchange [UK]	Taurus stock settlement system canceled after $600 million** is spent.
1993	Allstate Insurance Co.	Office automation system abandoned after deployment, costing $130 million.
1993	London Ambulance Service [UK]	Dispatch system canceled in 1990 at $11.25 million**; second attempt abandoned after deployment, costing $15 million.**
1993	Greyhound Lines Inc.	Bus reservation system crashes repeatedly upon introduction, contributing to revenue loss of $61 million.
1992	Budget Rent-A-Car, Hilton Hotels, Marriott International, and AMR [American Airlines]	Travel reservation system canceled after $165 million is spent.

Nota: Errores de software, adaptado de grandes errores producidos por software de mala calidad 2021. Charette, 2005. https://spectrum.ieee.org/why-software-fails

Para poder tener una idea del concepto de ingeniería de software, se la puede representar como lo que muestra la Figura 2.

Figura 2

¿Qué es la Ingeniería de Software?

Nota: Ingeniería de Software adaptado de IngSistemas, 2017. https://ing-sistemas.com/2017/02/09/que-es-la-ingenieria-software/

En un intento por definir ingeniería de software, varios autores han formulado sus definiciones; se citan algunas de ellas (IngSistemas, 2017):

"La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software" (Potts, 1993).

> Ingeniería del Software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software. (García, 2018)

"La ingeniería de software es el establecimiento de principios fundamentales de la ingeniería con objeto de desarrollar, en forma económica, software que sea confiable y que trabaje con eficiencia en máquinas reales". (Bauer, 1968)

"Ingeniería de software trata del establecimiento de los principios y de los métodos de la ingeniería, a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales" (Bauer, 1968).

Diferencias entre la ingeniería de software y las ciencias de la computación

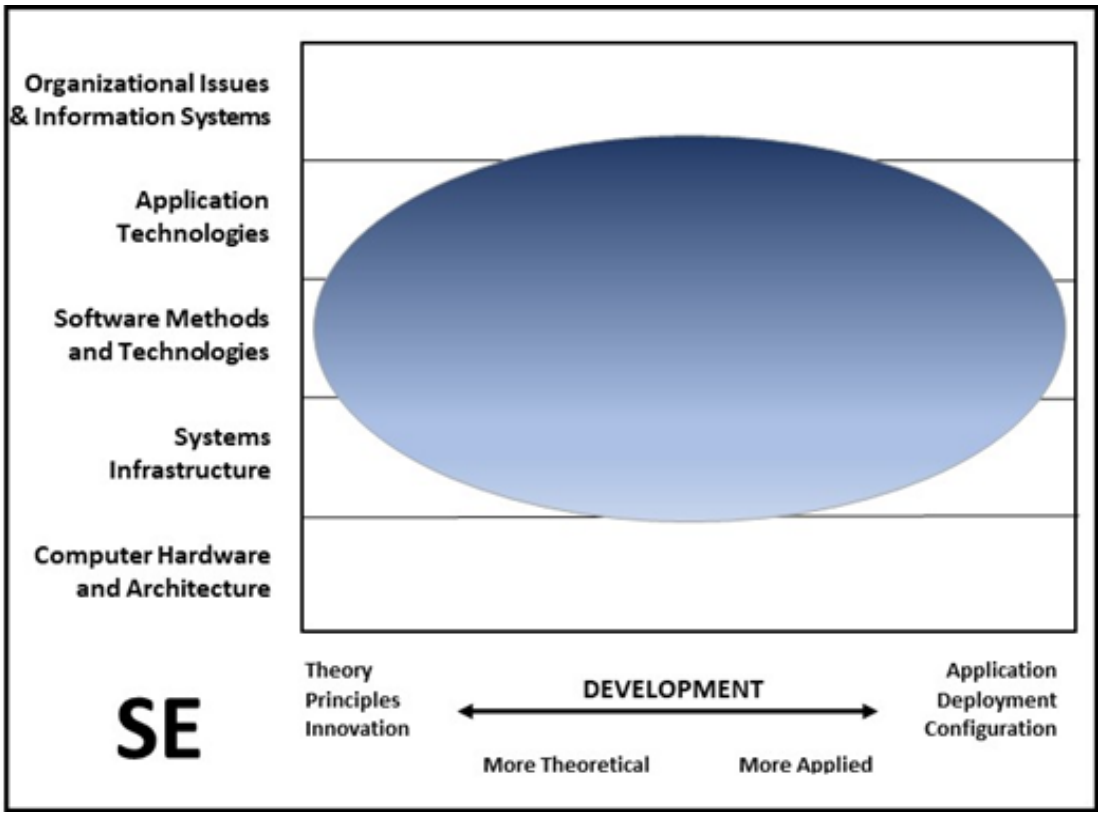
De acuerdo con el informe de la ACM (The Association for Computing Machinery, 2020), la AIS (Association of Information Systems) y la IEEE-CS (The Computer Society), en los años anteriores a 1990, la ingeniería informática surgida de la ingeniería eléctrica versaba sobre las cuestiones hardware y software relativas al diseño de dispositivos digitales. Durante la misma época, aquellos que querían llegar a ser expertos en el desarrollo de software debían estudiar Ciencias de la Computación.

El mundo pos 1990 presenta opciones con más significado. En ingeniería de computadores (Computer Engineering, CE), la atención del software se centra en los dispositivos hardware; para la ingeniería del software (Software Engineering, SE), el énfasis está en la creación de software que satisfaga las robustas necesidades del mundo real, y en la Computación (Computer Science, CS), el software es el "lenguaje" en que se expresan las ideas y donde se explora una amplia gama de problemas de computación y sus aplicaciones.

La ingeniería de software abarca toda la dimensión horizontal en el medio del diagrama, porque el tema cubre una amplia gama de aplicaciones de software a gran escala con respecto al desarrollo sistemático de software, como se indica en la Figura 3.

Figura 3

SE A computing curricula series report 2020 December 31

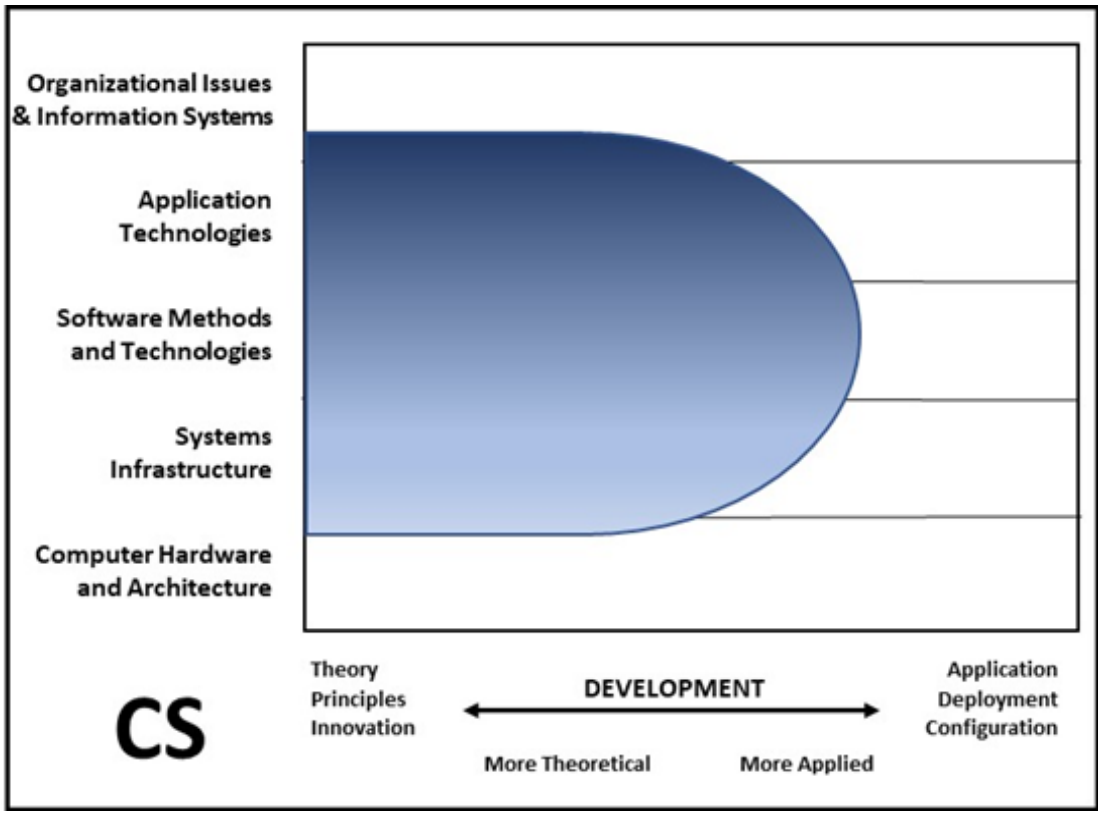


Nota: Adaptado de Currículo de Ingeniería de Sistemas de a ACM 2022 https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf

La computación, por su parte, abarca un amplio espectro, desde sus fundamentos teóricos y algorítmicos hasta las últimas novedades en robótica, visión por ordenador, sistemas inteligentes, bioinformática y otras áreas interesantes que pueden dividirse en tres categorías básicas: diseño e implementación de software; diseño de nuevas formas de utilizar los ordenadores, y desarrollo de formas efectivas de resolver problemas de computación. Esto se visualiza en la Figura 4.

Figura 4

CS A computing curricula series report 2020 December 31



Nota: Adaptado de Currículo de Ciencias de la computación de la ACM 2022. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf

Recursos complementarios 1

Por favor para consulta el siguiente enlace para reforzar los conocimientos adquiridos:

Accede aquí:

Actividad de aprendizaje 1

Descripción de la actividad

En el capítulo 1 de Fundamentos de la Ingeniería de Software, se aborda una introducción al origen, evolución y diferencias con las ciencias de la computación y otras, acorde a un estudio de la ACM de 2020.

Resuelva las diez preguntas que se muestran a continuación, donde las definiciones, características y diferencias con las otras ciencias se presentan. Puede encontrar material de apoyo en el siguiente enlace:

Descargar el "Chapter 1: Introduction (PPTX)"

1. ¿Qué es ingeniería de software?
2. ¿Cuáles son las actividades fundamentales de la ingeniería de software?
3. ¿Cuál es la diferencia entre ingeniería de software y ciencias de la computación?
4. ¿Qué es software?
5. ¿Cuáles son los atributos del buen software?
6. ¿Cuáles son los principales retos que enfrenta la ingeniería de software?
7. ¿Cuáles son los costos de la ingeniería de software?
8. ¿Cuáles son los mejores métodos y técnicas de la ingeniería de software?
9. ¿Qué entiende por paradigmas de la ingeniería de software?
10. ¿Cuál es la definición del modelo incremental versus otros paradigmas? Características y diferencias con otras ingenierías.

Autoevaluación del capítulo 1

1. **La persona que acuñó por primera vez el término "ingeniería del software" fue:**

   Margaret Hamilton.

   Margaret Sanger.

   Margaret Atwood.

2. **Los elementos que componen el software son:**

   Personal, proceso y producto.

   Programas, procedimientos, documentación y datos relacionados.

   Programas o instrucciones, partes y piezas y datos.

3. **Oficialmente, el término ingeniería del software se acuñó en:**

   La Conferencia de la OTAN de 1968.

   La Conferencia de la CEPAL de 1963.

   La Conferencia de la OTAN de 1986.

4. **La definición de tipo de software correcto es:**

   Programas que resuelven necesidades específicas de las organizaciones (software de sistemas).

   Conjunto de programas que han sido escritos para servir a otros programas (software de gestión o aplicación).

   Software que hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo (software de inteligencia artificial).

5. **¿Cuáles son los atributos de un buen software?**

   Funcionalidad y el rendimiento requerido por el usuario.

   Hacer que se malgasten los recursos del sistema.

   Mantenible, confiable y fácil de utilizar.

6. **Las características del software son:**

   El software usa componentes estándar con funciones e interfaces bien definidas.

El software se desarrolla o modifica con intelecto, no se fabrica en el sentido clásico.

El software se desgasta con el transcurso del tiempo.

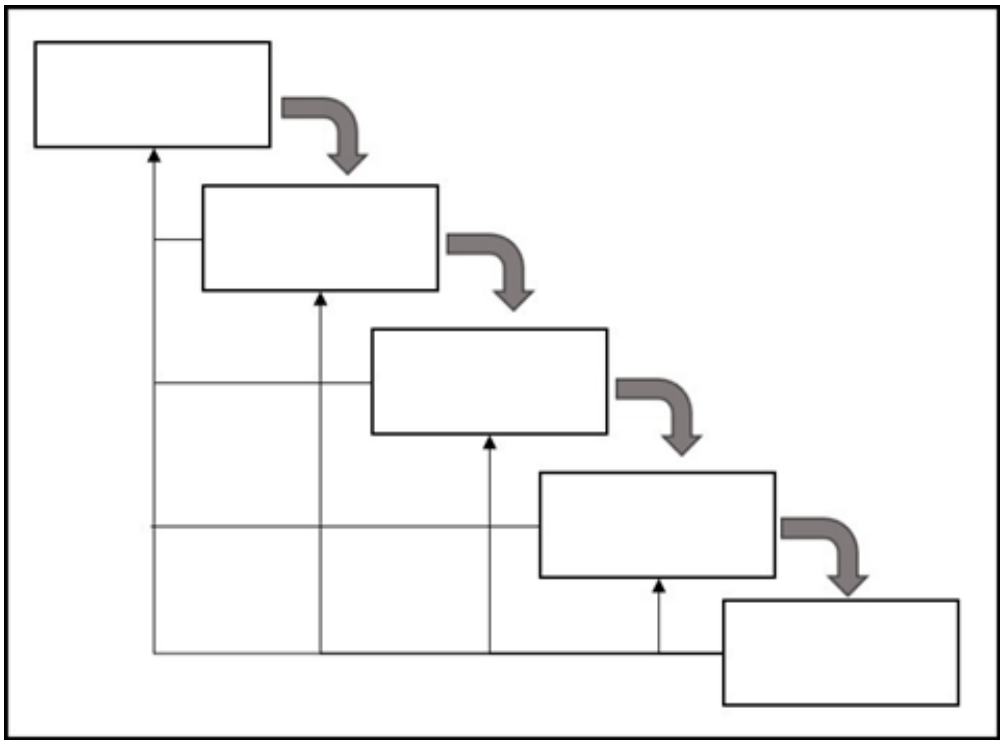
7. **La crisis del software se refiere a los problemas que desde sus inicios ha ido experimentado este. Muchas veces los problemas de gran magnitud se generan debido a la mínima eficacia que presenta una gran cantidad de empresas al momento de realizar un software.**

   Verdadero.

   Falso.

   Ninguna de las opciones.

8. **A partir del siguiente gráfico, los nombres de las fases del modelo en Cascada (Waterfall) son:**



   Gestión de proyecto, comunicación, planificación, diseño-modelado, construcción-implementación.

   Comunicación, planificación, diseño-modelado, construcción-implementación, entrega-implantación.

   Gestión de la configuración, comunicación, planificación, diseño-modelado, entrega-implantación.

9. **El modelo de proceso de software en espiral propuesto por Bohem conjuga la naturaleza iterativa de la construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. La etapa que no pertenece al modelo es:**

   Evaluación del cliente.

   Comunicación con el cliente.

   Definición de un paradigma de desarrollo.

10. **Se construye un buen sistema de información considerando que el punto de partida es:**

    La definición de requisitos claros es parte del proceso, pero no es del todo importante.

    Utilizar un proceso definido con fases claras, donde cada una de estas genera un producto final.

    Utilizar herramientas de desarrollo como medio para alcanzar un producto de calidad.

CAPÍTULO II

Producto Software

Definiciones

Un producto software es el resultado de un proceso de desarrollo, conocido también como ciclo de vida, cuyas actividades están debidamente estandarizadas, secuenciadas y estructuradas, de manera que el producto final cumpla con los requisitos funcionales y no funcionales solicitados por la parte interesada.

Las actividades comprendidas en este proceso incluyen, entre otras, el levantamiento de requisitos, sean estos funcionales o no funcionales, la elaboración del código fuente, pruebas, etc., hasta la entrega del producto funcional. Este desarrollo desde cero se lo ejecuta en un lenguaje de programación estructurado, orientado a objetos como Java, C,o Pyton (Somerville, 2011).

Características

Para definir un producto software, es necesario conocer en primer término, qué es un lenguaje de programación. (IngSistemas, 2017), como se puede apreciar en la siguiente figura.

Figura 5

¿Con qué lenguaje de programación puede empezar, que debo conocer sobre ellos?

Nota. 21 términos básicos de programación que debes dominar, adaptado de 10. Debugging. https://weremote.net/terminos-basicos-programacion/

Un lenguaje de programación no es más que un conjunto de códigos que se escriben bajo ciertas reglas, que le permiten a una persona escribir o asignar instrucciones en forma de algoritmos para obtener una respuesta lógica o física de un sistema informático.

Con una simple búsqueda en Google, usted puede encontrar enlaces a información relacionada con C, C++, Java, Python, Swift, etc., todos son lenguajes de programación y cada uno tiene características propias y enfoques distintos. La respuesta a la pregunta sobre qué lenguaje de programación es mejor, es bastante simple, depende de qué quiera hacer.

De este modo, desarrollar una aplicación para un iPhone SE es diferente que hacerlo para un teléfono móvil con sistema operativo Android. No es lo mismo trabajar en una aplicación para teléfonos móviles que desarrollar un videojuego para Xbox o trabajar en todo el software ERP de gestión.

Es recomendable iniciar aprendiendo C y luego enfrentar lenguajes de más alto nivel como Java o Python, cuya demanda es bastante alta. Quien conozca el lenguaje C estructurado encontrará una gran facilidad para aprender e interpretar otros lenguajes como C++, Java o Python (IngSistemas, 2017).

Tipos y clases de software

Según Pressman, en la actualidad hay siete categorías de software.

1. De sistemas

Conjunto de programas que han sido escritos para servir a otros programas, por ejemplo, compiladores, editores y herramientas para gestión de archivos que procesan estructuras de información complejas pero determinadas. (Pressman, 2005)

2. De aplicación

A este grupo pertenecen aquellos que manejan procesos de información comercial que, dicho sea de paso, constituye la mayor área de aplicación del software. Los sistemas discretos, por ejemplo, nóminas, cuentas de haberes-débitos, inventarios han evolucionado hacia el software de sistemas de información de gestión (SIG), que accede a una o más bases de datos que contienen información comercial relacionada.

3. De Ingeniería y ciencias

Estos se caracterizan por los algoritmos de manejo de números. Las aplicaciones son muy variadas, van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de los lanzamientos espaciales y desde la biología molecular a la fabricación automática. No obstante, las nuevas aplicaciones de este tipo se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (CAD), la simulación de sistemas y otras aplicaciones interactivas han comenzado a tomar características del software de tiempo real e incluso de sistemas.

4. Incrustado (empotrado)

Aquí encontramos a los productos inteligentes, que se han convertido en algo común en casi todos los mercados de consumo e industriales. Estos productos residen en la memoria de solo lectura y se utilizan para controlar productos, servicios y sistemas en diferentes ámbitos en los mercados antes indicados. Pueden ejecutar funciones muy limitadas y curiosas, por ejemplo, el control de las teclas de un horno microondas o suministrar una función significativa y con capacidad de control, como las que encontramos en las funciones digitales en un automóvil, como control de la gasolina, indicadores en el salpicadero, sistemas de frenado, entre otras.

5. Línea de productos

Son diseñados para proporcionar una capacidad específica para uso de muchos consumidores diferentes. Se centran en algún mercado particular, por ejemplo, control del inventario de productos o se dirige a mercados masivos de consumidores, como procesamiento de textos, hojas de cálculo, gráficas por computadora, multimedios, entretenimiento, administración de base de datos y aplicaciones para finanzas personales o de negocios.

6. Aplicaciones Web

Esta categoría de software centrada en la red abarca una amplia gama de aplicaciones y comprende tanto las basadas en navegador como las que residen en dispositivos móviles.

7. Inteligencia artificial (AI)

Este grupo hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo. Aquí encontramos a los sistemas expertos, también llamados sistemas basados en el conocimiento, por ejemplo, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas y los juegos.

Atributos de calidad del producto software

Definición de calidad de software

"La calidad del software es el grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario" (López Echeverry & Valencia Ayala, 2008).

"Un proceso de software efectivo aplicado de una manera que crea un producto útil que proporciona un valor medible para quienes lo producen y quienes lo utilizan" (Pressman, 2005).

La calidad del software se logra mediante la aplicación de métodos de ingeniería de software, prácticas de gestión sólidas y control de calidad integral.

La calidad del software no solo trata de si la funcionalidad de este se implementó correctamente, sino también, del grado de cumplimiento de los requisitos no funcionales del sistema. Ya en 1978, Boehm y sus colaboradores indicaban que existen quince importantes atributos de calidad de software. Por lo general, se considera que los atributos de confiabilidad son los atributos de calidad más importantes de un sistema; sin embargo, también es significativo el rendimiento del software. Como se muestra en la Figura 6, existe diversidad en los atributos de esta No Funcionalidad, llamados también características del software.

Figura 6

Atributos de calidad que son probados

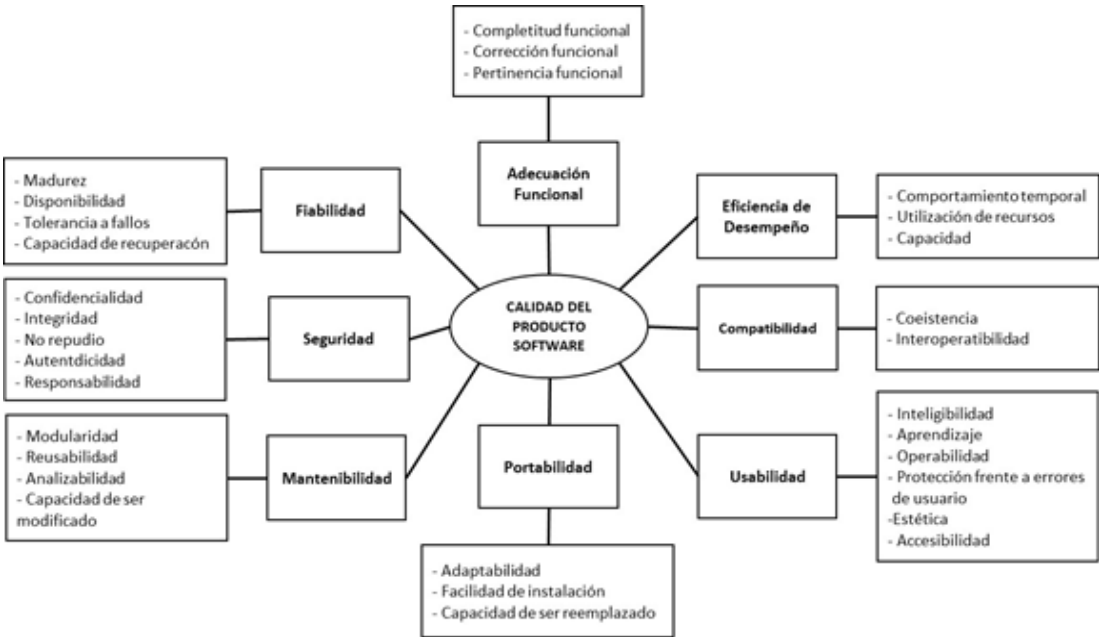
Nota. Pruebas para atributos de calidad, ISO/IEC 25010. adaptado de Testin IT https://www.testingit.com.mx/blog/atributos-de-calidad-de-software

La calidad del producto software se puede describir como el grado en el que dicho producto satisface los requisitos de sus usuarios, aportando de esta manera un valor. Son los requisitos no funcionales o de calidad, por ejemplo, funcionalidad, rendimiento, seguridad, mantenibilidad, los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas.

El modelo de calidad del producto definido por la ISO/IEC 25010 se encuentra compuesto por las ocho características de calidad, cada una con sus subcaracterísticas. En Figura 7 se presenta un esquema de este modelo.

Figura 7

Modelo de calidad ISO25010



Nota. Atributos de calidad de un producto de software. Adaptado de Curso Profesional de Arquitectura de Software. https://platzi.com/tutoriales/1248-pro-arquitectura/5498-atributos-de-calidad-de-un-producto-de-software/

Recursos complementarios 2

Consulte el siguiente enlace para reforzar los conocimientos adquiridos:

Accede aquí:

Actividad de aprendizaje 2

Responder las siguientes preguntas:

1. ¿Qué hay en la caja?
2. ¿Cuál es el trabajo que tenemos que hacer para lograr la meta?
3. ¿Qué tareas debemos realizar?

4. ¿Cómo nos aseguramos de que haremos un gran trabajo?
5. ¿Cómo medimos la calidad de lo que hemos hecho?
6. ¿Cómo nos aseguramos de que lo que hemos hecho esté realmente a la altura de las necesidades y solicitudes iniciales?
7. ¿Cómo minimizamos el esfuerzo para lograr la meta?
8. ¿Qué habilidades necesitamos?
9. ¿Qué herramientas necesitamos?
10. ¿Cuántas personas se necesitan?
11. ¿Qué habilidades, perfiles, experiencia?

Autoevaluación del capítulo 2

1. **¿Qué es una metodología de desarrollo de software?**

   Un conjunto de lenguajes de programación que permiten analizar, diseñar y construir productos software.

   Un conjunto de rutinas de programación que permiten desarrollar aplicaciones de forma ágil.

   Un conjunto de métodos que cubren todo el ciclo de vida de desarrollo de sistemas y que están unidos por un enfoque general filosófico.

2. **Las actividades estructurales del proceso de desarrollo de software son:**

   Despliegue/comunicación/modelado/diseño.

   Gestión de la configuración/modelado.

   Administración del proyecto, diseño.

3. **Una de las razones por la cual se incrementan los costos al agregar funcionalidad después que un sistema está en operación es:**

   Lenguaje de programación seleccionado.

   Edad de los programadores.

   Práctica de desarrollo deficiente.

4. **Ingeniería de software asistida por computador significa:**

   Conjunto de herramientas de software integradas para ser utilizadas por otras.

Conjunto de herramientas de hardware integradas para ser utilizadas por otras.

Elementos físicos del computador integrados para ser utilizados por otras.

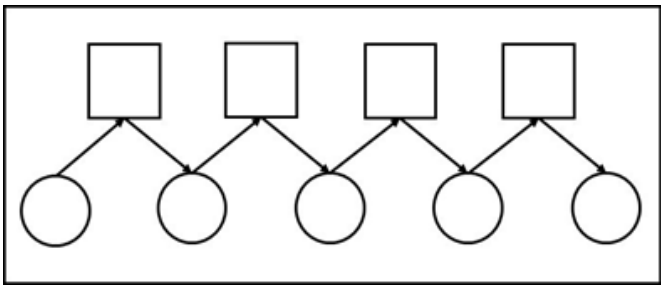
5. **La metodología con sus tipos correspondientes de clasificación es:**

   Metodología Scrum (estructurada).

   Metodología XP (orientada a objetos-ágil).

   Proceso unificado de desarrollo (estructurada).

6. **En el gráfico propuesto, la secuencia ordenada de las figuras es:**



   Necesidad, obtener requisitos, diseñar sistema, especificación de requisitos, diseño, codificar, código, probar, sistema de software.

   Obtener requisitos, diseñar sistema, especificación de requisitos, diseño, codificar, código, probar, sistema de software, necesidad.

   Necesidad, especificación de requisitos obtener requisitos, diseñar sistema, diseño, codificar, código, probar, sistema de software.

7. **Los modelos de calidad del producto software y los modelos de calidad del proceso de desarrollo de Software son:**

   ISO/IEC 25010 producto software.

   ISO/IEC 20000 producto software.

   ISO 9126 proceso de desarrollo.

8. **La norma ISO/IEC 25010 identifica características y subcaracterísticas de calidad del producto software. La opción correcta es:**

   Seguridad es subcaracterística.

   Eficiencia de desempeño es característica.

   Fiabilidad es subcaracterística.

CAPÍTULO III

Proceso de Desarrollo

Fases de un proceso genérico de desarrollo de software

"Un proceso software es una serie de actividades relacionadas que conducen a la elaboración de un producto software". (Somerville, 2011)

Existen varios procesos de software, pero todos deben incluir cuatro actividades básicas requeridas por la ingeniería de software: especificación, diseño e implementación, validación y evolución (Somerville, 2011).

La especificación del software define la funcionalidad y las restricciones. El diseño e implementación del software se desarrolla para cumplir con las especificaciones. La validación del software examina el software para asegurar el cumplimiento de requerimientos y especificaciones establecidos por el cliente. La evolución del software es una característica de flexibilidad que le permite al software transformarse para satisfacer las necesidades cambiantes del cliente.

De la misma forma como existen actividades, también existen descripciones de los procesos software, estas son: productos resultantes de la actividad del proceso, por ejemplo, el resultado de la especificación de requisitos de software (ERS) es el documento de ERS; roles son las responsabilidades de las personas que intervienen en el proceso, por ejemplo, analista, programador, diseñador, tester e ingeniero de requisitos; precondiciones y postcondiciones son declaraciones válidas antes y después de que se ejecute una actividad del proceso, por ejemplo, antes de iniciar el diseño, la precondición es la aprobación de los requisitos por parte del cliente, una postcondición será la revisión de los modelos UML que se describen en la arquitectura.

Ciclos de vida del proceso de desarrollo de software

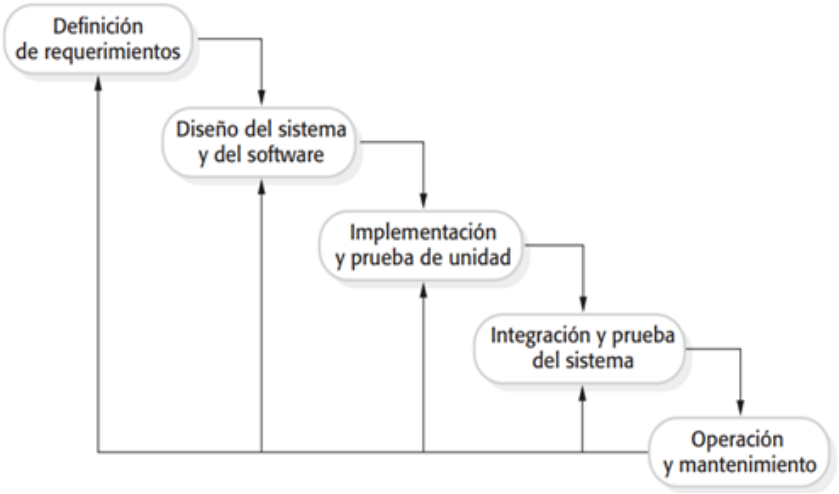
A continuación, presentaremos algunos modelos generales de proceso también llamados "paradigmas de proceso", describiéndolos desde su arquitectura sin entrar en detalles.

Modelo en cascada (waterfall)

Ocupa las actividades importantes del proceso de especificación, desarrollo, validación y evolución, luego los representa como fases separadas del proceso, por ejemplo, especificación de requisitos, diseño de software, implementación, pruebas, etc. En la Figura 8 se indica este modelo.

Figura 8

Modelo en cascada



Nota. Modelo en cascada. adaptado de Procesos de software Unidad 2 - Software Enginnering - Ian Sommerville, https://es.slideshare.net/MatiasGonzaloAcosta/procesos-de-software-unidad-2-software-enginnering-ian-sommerville

Desarrollo incremental

Vincula las actividades de especificación, desarrollo y validación. Para el desarrollo del sistema se realiza una serie de versiones (incrementos) y cada versión añade funcionalidad a la versión anterior, como se indica en la Figura 9.

Figura 9

Desarrollo incremental

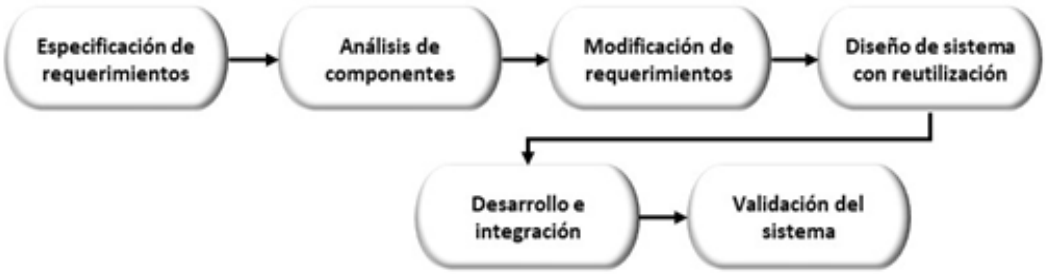
Nota. Desarrollo incremental. adaptado de Procesos de software Unidad 2 - Software Enginnering - Ian Sommerville.https://es.slideshare.net/MatiasGonzaloAcosta/procesos-de-software-unidad-2-software-enginnering-ian-sommerville

Ingeniería de software

Reutiliza un número importante de componentes, el proceso de desarrollo del sistema se enfoca en la integración de estos en un sistema evitando desarrollarlos desde cero como se indica en la Figura 10.

Figura 10

Ingeniería de software orientada a la reutilización



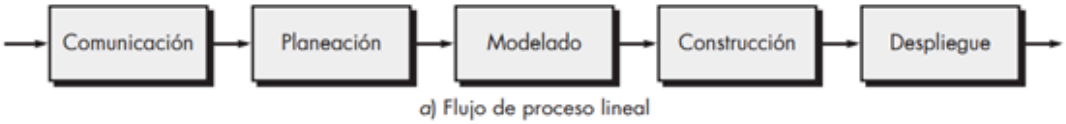
Nota. Proceso de reutilización en la Ingeniería de Software Unidad 2 - Software Enginnering - Ian Sommerville. https://es.slideshare.net/MatiasGonzaloAcosta/procesos-de-software-unidad-2-software-enginnering-ian-sommerville

Flujos del proceso de desarrollo de software

El flujo del proceso de desarrollo de software caracteriza la manera en que están organizadas las actividades estructurales, las acciones y las tareas que ocurren dentro de cada una con respecto a la secuencia y al tiempo, tal como se indica en la Figura 11, en la Figura 12, en la Figura 13 y en la Figura 14.

Figura 11

Flujo del proceso lineal

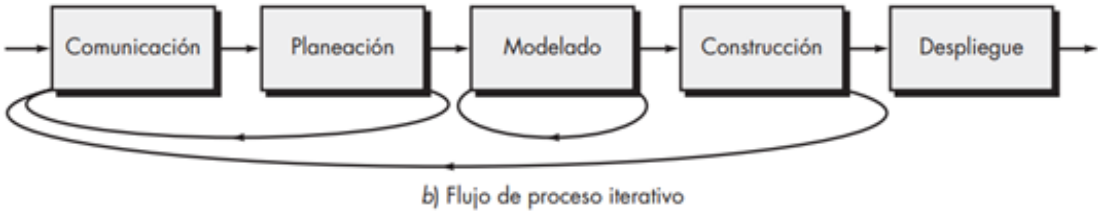


a) Flujo de proceso lineal

Nota. Procesos de desarrollo de Software. Adaptado de Ramiro Estigarribia. Canesehttps://medium.com/@ramiroec/procesos-de-desarrollo-de-software-d06d18ebd78f

Figura 12

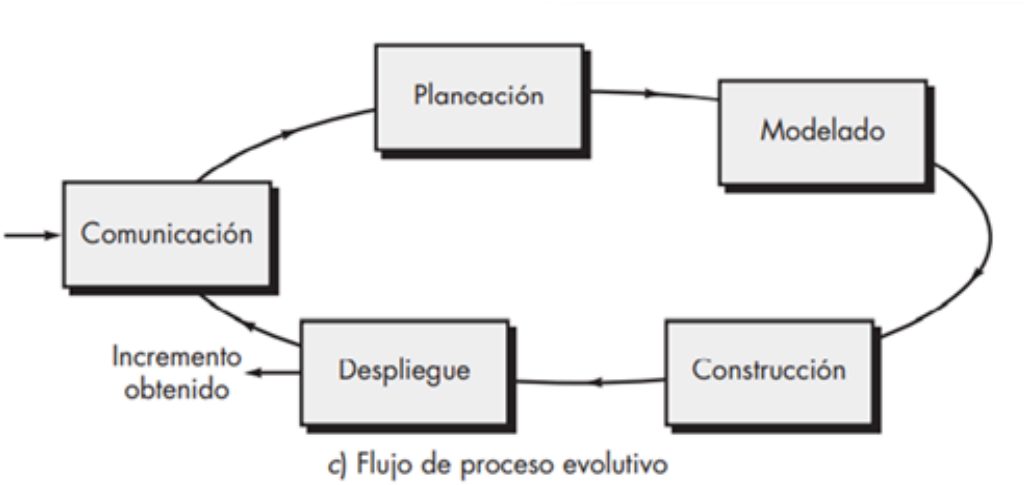
Flujo del proceso iterativo



Nota. Procesos de desarrollo de Software. Adaptado de Ramiro Estigarribia. Canesehttps://medium.com/@ramiroec/procesos-de-desarrollo-de-software-d06d18ebd78f

Figura 13

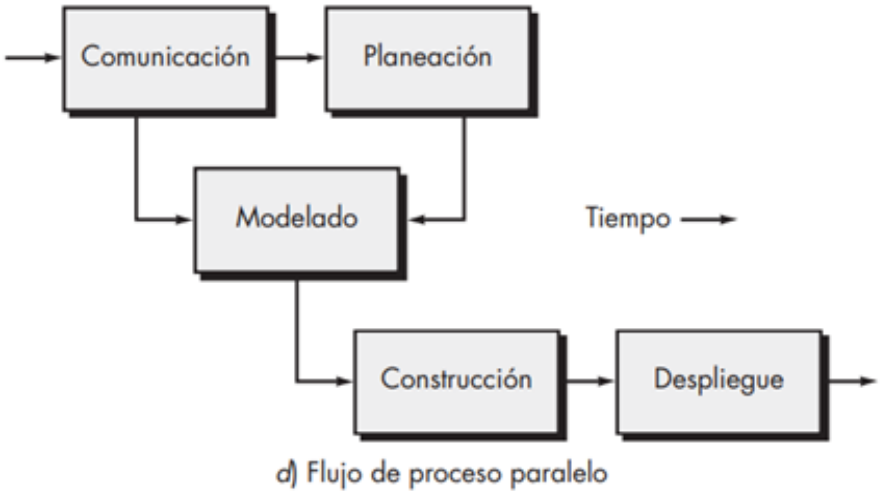
Flujo de proceso evolutivo



Nota. Procesos de desarrollo de Software. Adaptado de Ramiro Estigarribia. Canesehttps://medium.com/@ramiroec/procesos-de-desarrollo-de-software-d06d18ebd78f

Figura 14

Flujo del proceso en paralelo



Nota. Procesos de desarrollo de Software. Adaptado de Ramiro Estigarribia. Canesehttps://medium.com/@ramiroec/procesos-de-desarrollo-de-software-d06d18ebd78f

Recursos complementarios 3

Consulte el siguiente enlace para reforzar los conocimientos adquiridos:

Accede aquí:

Actividad de aprendizaje 3

Responda las siguientes preguntas:

1. **Seleccione una o más de una de la siguiente afirmación: Identifique los tipos de clases de metodologías de desarrollo de software.**

   Estructurada

   Orientada a objetos

   Espiral

Diseño rápido de aplicaciones.

2. **Identifique la definición de metodología de desarrollo de software**

   Un conjunto de aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero.

   Un conjunto de técnicas y procedimientos organizados en fases para el desarrollo de productos software, de manera eficaz, y abarca el ciclo de vida del mismo.

   Un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

   Ninguna de las anteriores.

3. **Complemente la siguiente afirmación: Las metodologías orientadas a objetos centran su análisis, diseño e implementación en…**

   El comportamiento y la estructura.

   Los procesos y los datos.

   La funcionalidad.

   Los objetos.

4. **En el siguiente listado identifique los principios del "manifiesto ágil"**

   Es más importante crear un producto software que funcione que escribir documentación exhaustiva.

   Entregar el producto software lo más pronto, sin realizar documentación y minimizar los costos.

   La colaboración con el cliente debe prevalecer sobre la negociación de contratos.

   Los objetos.

5. **Empareje las definiciones de los diagramas de UML con los términos correspondientes**

   1D,2C,3B,4A

   1A,2B,3C,4D

   1B,2A,3D,4C

   1C,2D,3A,4B

6. **Seleccione una opción de la siguiente afirmación: Las fases del proceso de ingeniería de requisitos son:**

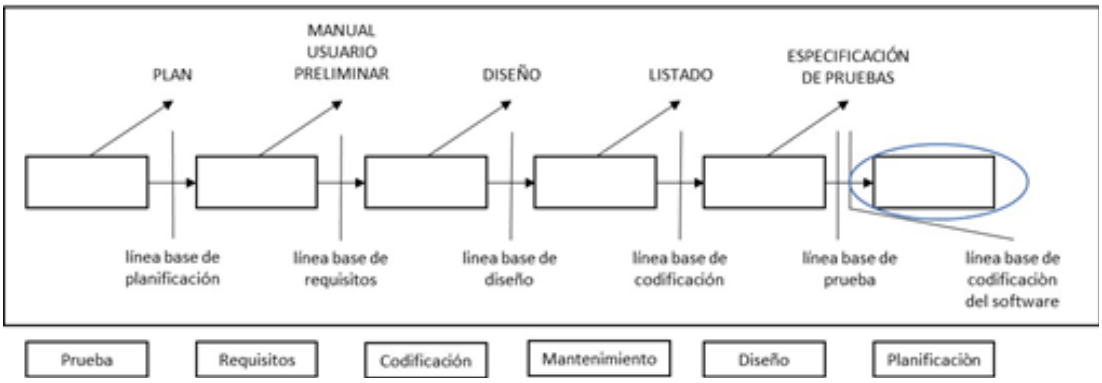
   Comunicación, planificación, modelado y desarrollo

   Educción, análisis / negociación, especificación y validación

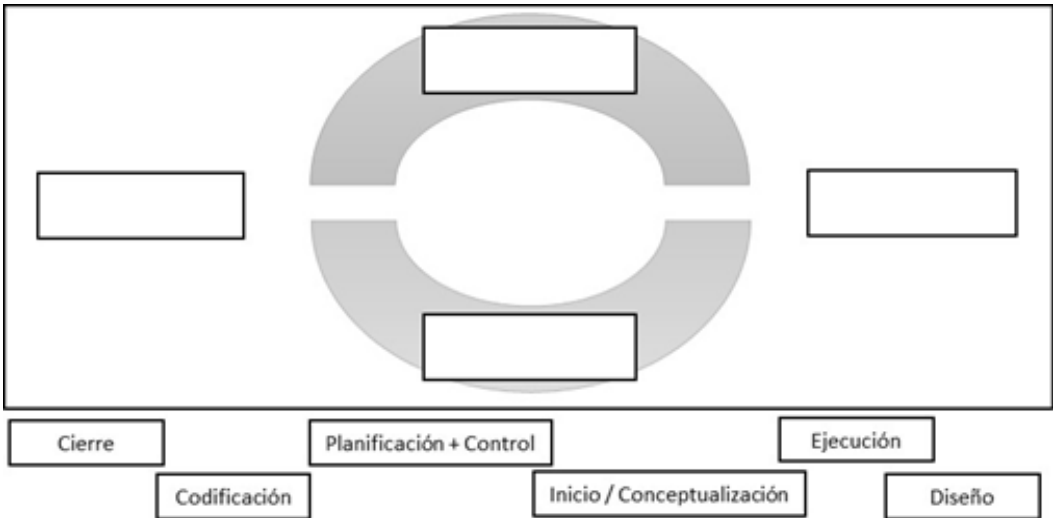
   Observación, planificación, reflexión, acción

   Ninguna de las anteriores

7. **Complete el siguiente gráfico que representa la línea base y los elementos de la configuración del software.**

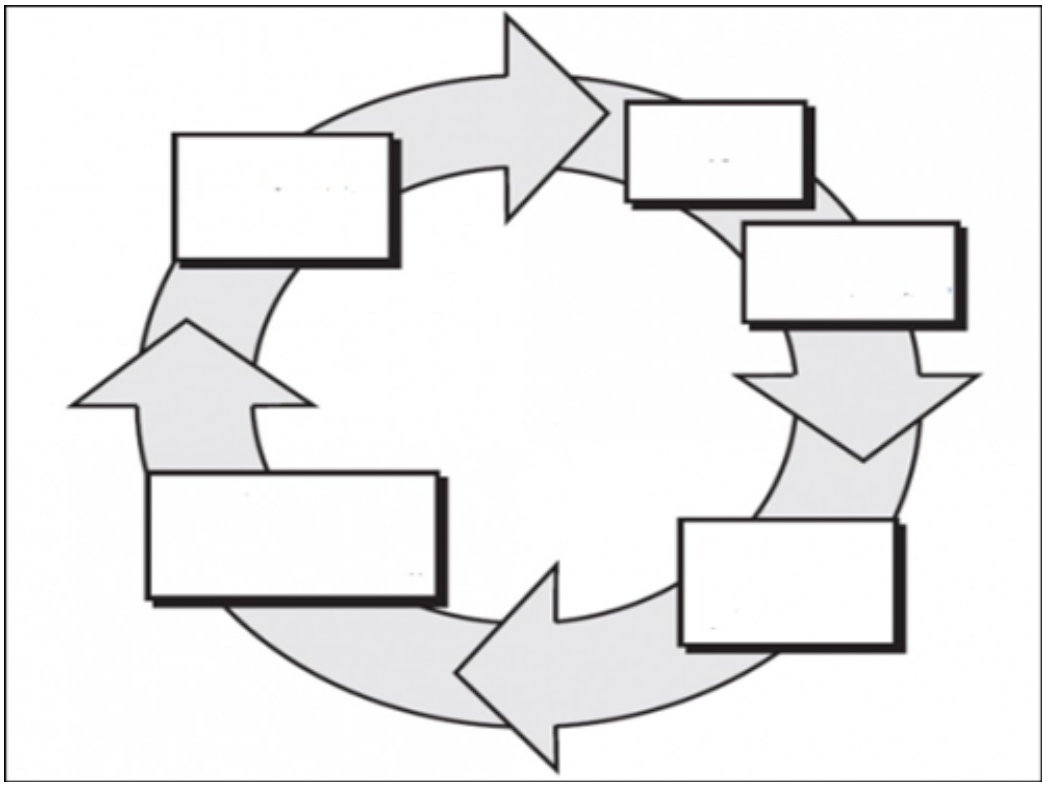


8. **El siguiente gráfico representa las fases de una metodología de gestión de proyecto de desarrollo de software, revisada en clases. Complete el gráfico con el nombre de las fases.**



Autoevaluación del capítulo 3

1. **En el siguiente gráfico, que representa el modelo de construcción de prototipos, los nombres de cada fase respetando el orden de ejecución son:**



Plan rápido, modelado de diseño rápido, construcción del prototipo despliegue, entrega y retroalimentación, comunicación, plan rápido.

Modelado de diseño rápido, construcción del prototipo, despliegue, entrega y retroalimentación, plan rápido, comunicación.

Comunicación, plan rápido, modelado de diseño rápido, construcción del prototipo despliegue, entrega y retroalimentación

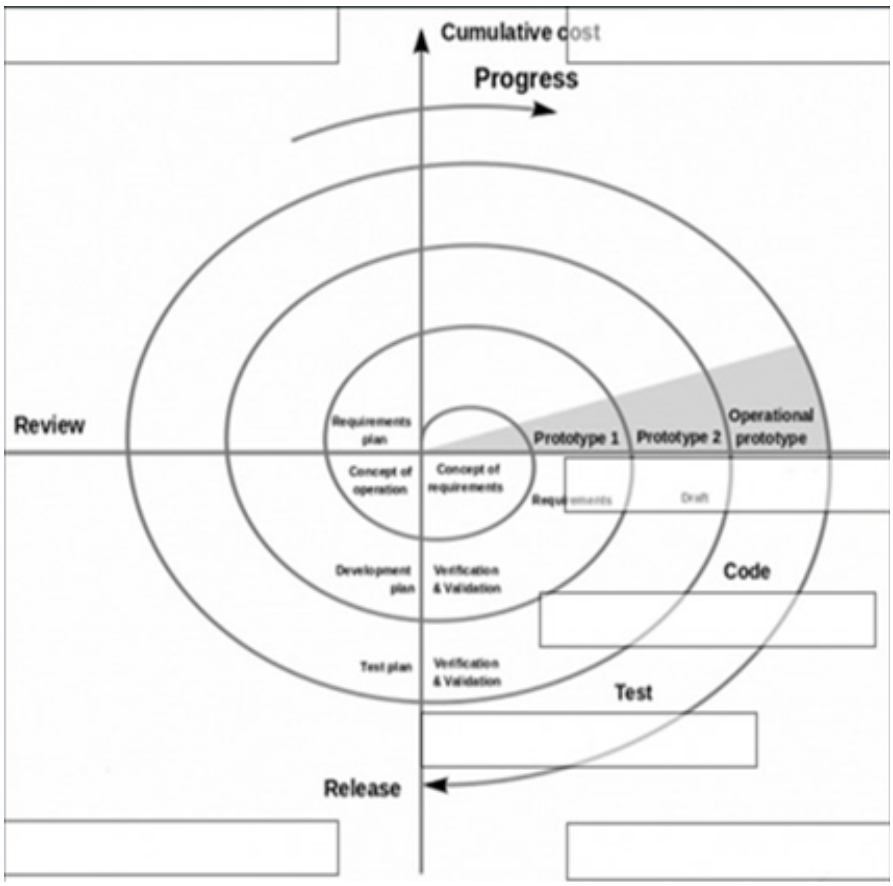
2. **¿Qué es un proceso de software?**

Representa los roles de las personas involucradas en el proceso de software y las actividades de las que son responsables.

Es un conjunto de actividades cuya meta es el desarrollo u optimización del software.

Es la secuencia de actividades en el proceso junto con sus entradas, salidas y dependencias, las actividades en este modelo representan acciones humanas.

3. **En el siguiente gráfico, que representa el modelo de espiral, los nombres de cada fase respetando el orden de ejecución van así:**

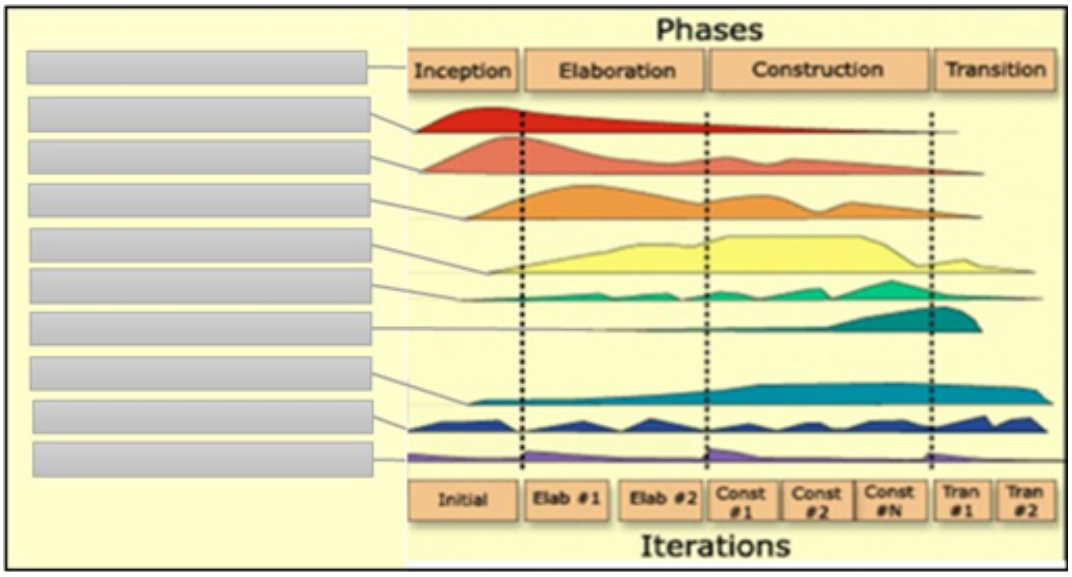


Determinar objetivos, identificar y resolver riesgos, diseño detallado, integración, implantación, desarrollo y pruebas, plan siguiente iteración.

Identificar y resolver riesgos, diseño detallado, integración, implantación, desarrollo y pruebas, plan siguiente iteración, determinar objetivos.

Determinar objetivos, diseño detallado, integración, implantación, desarrollo y pruebas, identificar y resolver riesgos, plan siguiente iteración.

4. **En el siguiente gráfico del proceso unificado de desarrollo, los nombres de cada fase respetando el orden de ejecución van así:**



Procesos estructurales: gestión de la configuración y cambios requisitos, análisis y diseño, implementación, prueba, entrega, procesos de soporte, modelamiento del negocio, gestión del proyecto, entorno.

Procesos estructurales: gestión del proyecto, requisitos, análisis y diseño, implementación, prueba, entrega, gestión de la configuración y cambios procesos de soporte, modelamiento del negocio, entorno.

Procesos estructurales: modelamiento del negocio, requisitos, análisis y diseño, implementación, prueba, entrega, procesos de soporte, gestión de la configuración y cambios, gestión del proyecto, entorno.

5. **¿Qué es la metodología de desarrollo de software?**

Es un conjunto de técnicas procedimientos organizados en fases para el desarrollo de productos software, de manera eficaz, y abarca el ciclo de vida del mismo. Es una colección de métodos para la resolución de una clase de problema.

Es un conjunto de técnicas ciclos de vida, en fases para el desarrollo de productos software, de manera eficaz, y abarca procedimientos organizados del mismo para la resolución de una clase de problema.

Es un conjunto de métodos para el desarrollo de productos software, de manera eficaz, y abarca el ciclo de vida del mismo. Es una colección de técnicas procedimientos organizados en fases para la resolución de una clase de problema.

6. **Ud. es el líder de un equipo de desarrollo de un aplicativo que necesita maximizar la precisión, el comportamiento del tiempo y la capacidad de prueba; así también, debe minimizar la ambigüedad, por tanto, decidirá un lenguaje de programación que se base en algún tipo de notación, es decir elegirá:**

   Notaciones formales.

   Notaciones lingüísticas.

   Notaciones visuales.

7. **Una de las causas que motivó la llamada crisis del software fue:**

   La falta de programadores altamente capacitados.

   Los cambios continuos pedidos por el cliente.

   Los errores debido a la baja calidad del software.

8. **Las fases del proceso unificado de desarrollo son:**

   Incepción, elaboración, construcción, transición.

   Comunicación, elaboración, construcción, transición.

   Modelamiento, análisis y diseño, construcción, pruebas y despliegue.

9. **El tipo de proceso del ciclo de vida del software (ISO 12207) es:**

   Operación, suministro, adquisición, procesos primarios.

   Documentación, resolución de problemas procesos organizacionales.

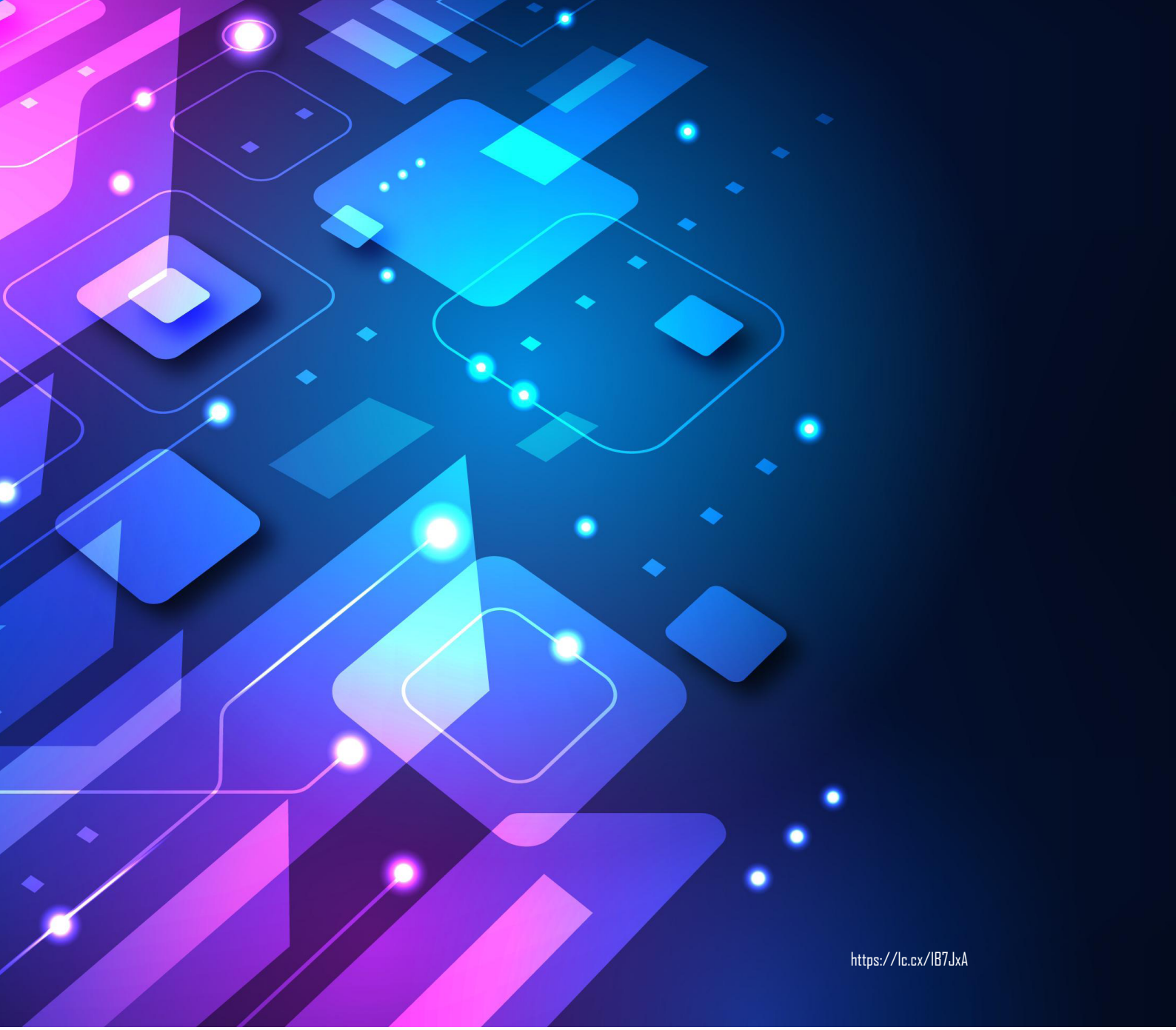
   Mejoras, entrenamiento, infraestructura, procesos de soporte.

10. **A partir de los requisitos del sistema levantados por el ingeniero de software, se obtiene como producto final la especificación del sistema que describe:**

    Los modelos del sistema que obedecen a los requisitos funcionales y no funcionales de los usuarios.

    La función y las características de un sistema de computación y las restricciones que gobiernan su desarrollo.

    La planificación detallada de las etapas de una metodología a seguir en el proceso de construcción del software.



CAPÍTULO IV

Fundamentos de la ingeniería de requisitos

Definiciones requisito de software e ingeniería de requisitos del software

De acuerdo con la Real Academia Española de la Lengua, es la "circunstancia o condición necesaria para algo" (Real Academia Española, 2020).

En informática, la definición de requisito no está consensuada y se pueden encontrar varias definiciones que incluyen a la IEEE, así:

(1) Una condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo. (2) Una condición o capacidad que debe tener un sistema o un componente de un sistema para satisfacer un contrato, una norma, una especificación u otro documento formal. (3) Una representación en forma de documento de una condición o capacidad como las expresadas en (1) o en (2) (IEEE Xplore, 2002).

Requisitos para un sistema

Son descripciones de lo que el sistema debe hacer, el servicio que ofrece y las restricciones en su operación, de tal forma que estos reflejen las necesidades de los clientes para un sistema que atiende este propósito. Por ejemplo, controlar un dispositivo, buscar o realizar un pedido de información, de ahí que la ingeniería de requisitos (IR) es el proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones (Somerville, 2011).

Los requisitos del usuario y del sistema se definen de la siguiente manera: **Del usuario,** son enunciados en un lenguaje natural junto con diagramas acerca de qué servicios esperan los usuarios del sistema y de las restricciones con las cuales este debe operar (Somerville, 2011).

Del sistema, son descripciones más detalladas de las funciones, servicios y las restricciones operacionales del sistema software. El documento de especificación de requisitos (ERS) tiene que definir con exactitud lo que se implementará en el sistema. (Somerville, 2011)

Stakeholder, como término tiene algunas traducciones, a saber: "partes interesadas", "involucradas", es decir, aquellas personas que tengan interés directo o indirecto en el sistema. Son la principal fuente de información para la e-licitación o educción de requisitos; por otro lado, son quienes interactúan

con el sistema, también son aquellos profesionales especializados que se involucran en el proceso acorde a la naturaleza del sistema (Young, 2004).

Existe una diversidad de definiciones de ingeniería de requisitos, se citan algunas de estas.

Para Hsia; Davis & Kung (1993), la ingeniería de requisitos se define como:

> …un proceso participativo, iterativo e incremental de construcción de los requisitos, en el que se avanza desde especificaciones iniciales, que no poseen las propiedades deseadas, hasta especificaciones finales que tienen características ideales, esto es: completas, acordadas entre los involucrados y documentadas con un nivel adecuado de formalidad. (Hinojosa, Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE, 2014)

Para Pohl (2010) la ingeniería de requisitos es:

> …el proceso de la obtención de las necesidades individuales y las necesidades de las partes interesadas, creando una documentación detallada de los requisitos acordados y especificados, de tal manera que puedan servir de base para todas las otras actividades de desarrollo del sistema

Para Hull; Jackson & Dick (2011), la ingeniería de requisitos es: "…el subconjunto de la Ingeniería de Sistemas que se ocupa del descubrimiento, desarrollo, seguimiento, análisis, clasificación, comunicación y gestión de requisitos que definen el sistema y los niveles sucesivos de abstracción" (Hinojosa, Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE, 2014).

La ingeniería de requisitos es un enfoque metódico, sistemático y disciplinado para la especificación y la gestión de requisitos, que tiene entre sus objetivos: conocer los requisitos importantes, asegurando un consenso con todos los stakeholders; documentarlos acorde a los estándares establecidos y priorizarlos; ordenarlos sistemáticamente, así como entender y documentar las necesidades de los implicados.

Proceso de la ingeniería de requisitos del software

Tipos de requisitos: funcionales y no funcionales

Un requisito funcional es un requerimiento relacionado con el resultado del comportamiento que debería ser provisto por una función del sistema (Pohl, K. & Rupp, C., 2015).

Un requisito de calidad define una característica que un sistema o sus funciones deben tener y, a menudo, influyen en la decisión de la arquitectura del sistema.

Una restricción es un requerimiento (de carácter técnico organizativo) que limita el enfoque de la solución.

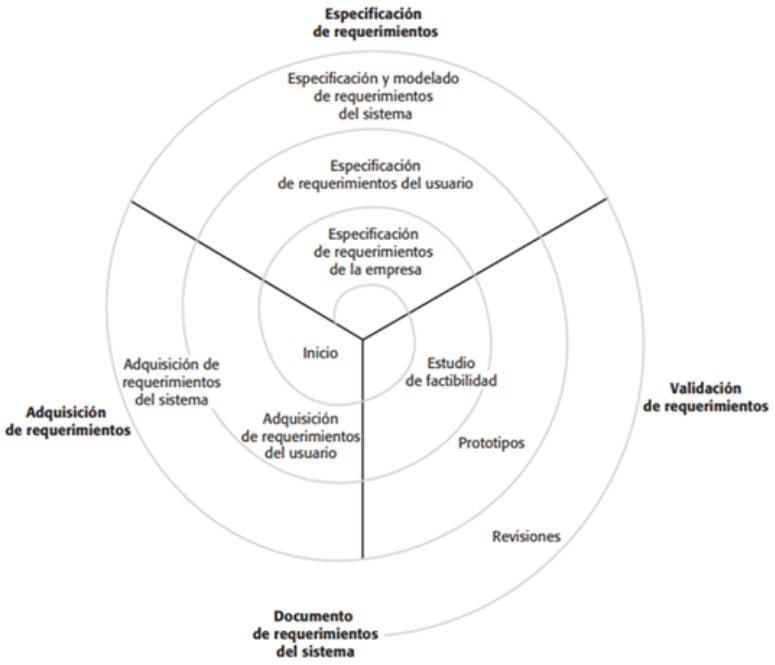
Los requisitos de calidad y las condiciones o restricciones también son llamados requisitos no funcionales.

Procesos que guían la ingeniería de requisitos

Los procesos que guían la ingeniería de requisitos incluyen cuatro actividades de alto nivel: adquisición, especificación, validación y documentación de requisitos, tal como se indica en la Figura 15.

Figura 15

Visión espiral del proceso de ingeniería de requisitos



Nota. Proceso de ingeniería de requisitos. Adaptado de capítulo 4 Ingeniería de Requerimientos Somerville. https://slideplayer.es/slide/17988345/

En el subproceso de adquisición y análisis encontramos las siguientes actividades: descubrimiento de requisitos, clasificación y organización, priorización y negociación, y especificación de requisitos, tal como se indica en la Figura 16.

Figura 16

Adquisición y análisis de requisitos

Nota. Procesos de ingeniería de requisitos. Adaptado de capítulo 4 Ingeniería de Requerimientos Somerville https://www.academia.edu/40713382/Libro_Somerville

Descubrimiento de requisitos Permite la interacción con los stakeholders del sistema para descubrirlos.

Clasificación y organización de requisitos. Provee la compilación no estructurada de estos.

Priorización y negociación. Permite resolver conflictos por participación de varias personas.

Especificación de requisitos. Genera documentación formal e informal. La primera basada en un estándar, por ejemplo, IEEE830 de 1998; la segunda, por

medio de un documento escrito a manera de historias de usuario con lenguaje natural.

La validación de requisitos es el proceso para verificar si los requisitos definen realmente el sistema que el cliente desea. Las comprobaciones a realizarse son: comprobaciones de validez, consistencia, totalidad, realismo y verificabilidad; adicionalmente, existen algunas técnicas de validación de requisitos que pueden usarse individualmente o en conjunto con otras como: revisiones de requisitos, creación de prototipos y generación de casos de prueba (Somerville, 2011).

Recursos complementarios 4

Visite el siguiente enlace:

Accede aquí:

Responda las siguientes preguntas:

1. **El modelo construcción de prototipos incluye actividades como: comunicación, planeación, modelado, construcción y despliegue.**

   Cierto

   Falso

   Ninguna de las opciones

2. **El modelo incremental llamado también DRA, es una adaptación a alta velocidad del modelo en cascada.**

   Cierto.

   Falso.

   Ninguna de las opciones

3. **El modelo incremental combina elementos del modelo en cascada aplicado en forma iterativa, se enfoca en la entrega de un producto operacional en cada incremento**

   Cierto.

   Falso.

   Ninguna de las opciones

4. **El modelo incremental ayuda al ingeniero de sistemas y al cliente entender de mejor manera cuál será el resultado de la construcción cuando los requisitos estén satisfechos**

   Cierto

   Falso

   Ninguna de las opciones

5. **Software que se caracteriza por el uso de algoritmos de manejo de números. Las aplicaciones van desde la astronomía a la vulcanología, dinámica orbital de las lanzaderas espaciales, biología molecular, entre otras.**

   Software de sistemas

   Software inteligencia artificial

   Software empotrado

   Software de gestión o aplicación

6. **Software que reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo**

   Software de sistemas

   Software inteligencia artificial

   Software empotrado

   Software de gestión o aplicación

7. **Conjunto de programas que han sido escritos para servir a otros programas**

   Software de sistemas

   Software inteligencia artificial

   Software empotrado

   Software de gestión o aplicación

8. **Programas que resuelven necesidades específicas de las organizaciones. Las aplicaciones en esta área reestructuran los datos existentes para facilitar las operaciones o gestionar la toma de decisiones en las empresas**

Software de sistemas

Software inteligencia artificial

Software empotrado

Software de gestión o aplicación

Actividades de aprendizaje 4

1. **Seleccione una o más de una opción: identifique las notaciones que permiten representar la vista dinámica o comportamiento del sistema.**
   - Diagramas de implementación
   - Tarjetas de clase-responsabilidad-colaboración (CRC)
   - Diagramas de comunicación
   - Diagramas de clase y objeto
   - Diagramas de actividad
   - Diagramas de secuencia
   - Diagramas de transición estados
   - Diagramas de componentes
2. **A partir de los requisitos del sistema levantados por el ingeniero se obtiene como producto final la especificación del sistema, que describe:**

   Los modelos del sistema que obedece a los requerimientos funcionales y no funcionales de los usuarios.

   La función y características de un sistema de computación y las restricciones que gobiernan su desarrollo.

   La lluvia de ideas que recoge los requerimientos del usuario, producto de las reuniones preliminares

   Ninguno de los anteriores
3. **Del siguiente listado de actividades, cuáles no son actividades centrales de la ingeniería de requisitos.**

   La validación de requisitos.

La formalización de requisitos.

La negociación de requisitos.

La documentación de requisitos

4. **Escoja la ventaja de utilizar cuestionarios para la educación de requisitos**

   Es posible llegar un alto número de participantes.

   Se pueden hacer afirmaciones estadística mente relevantes con respecto a los requisitos.

   Los cuestionarios ayudan a comprender mejor a los insatisfactores (factores básicos).

   Los cuestionarios permiten validar la comprensión de los participantes

5. **Respecto de la reparación de fallas, indique en qué fase es más costoso realizar las reparaciones o correcciones, en caso de detectar errores**

   En la fase de ingeniería de requisitos.

   En la fase de operación del sistema.

   En la fase de programación

   En la fase de diseño

6. **Según la Norma IEEE 830, los requerimientos deben cumplir con varios criterios de calidad. En la siguiente lista selecciones los criterios correspondientes.**

   Consistente

   Extenso

   Lenguaje altamente especializado

   Evaluado

   Correcto

   Ambiguo

7. **Usted es el líder del equipo de desarrollo de un aplicativo que requiere maximizar la precisión, el comportamiento del tiempo y la capacidad de prueba; así también debe minimizar la ambigüedad. Por lo tanto, usted decidirá un lenguaje de programación que se base en qué tipo de notaciones.**

Notaciones formales.

Notaciones lingüísticas

Notaciones visuales

Ninguno de los anteriores

8. **Se pueden utilizar plantillas de frases para la documentación de requisitos en lenguaje natural. Usted desea introducir este tipo de plantillas de frases en su proyecto y tiene que convencer a su jefe de proyecto de sus ventajas. ¿Qué argumentos usted podría utilizar mejor en este debate? (2 respuestas)**

   Aprender a escribir requisitos de acuerdo con las plantillas de frases no requiere mucho tiempo

   Un requisito redactado de acuerdo con una plantilla de frase satisface todos los criterios de calidad para los requisitos

   La alta calidad de los requisitos empieza por la documentación inicial.

   Los requisitos formulados de acuerdo con la plantilla de frases no contienen ningún efecto propio de una transformación lingüística

Autoevaluación del capítulo 4

1. **Un requisito funcional es:**

   El sistema interactuará con el sistema de entrega de pedidos.

   El sistema permitirá crear reservaciones de las cotizaciones.

   El sistema permitirá obtener respaldos de las transacciones cada hora.

2. **¿Cuál es el propósito del análisis de requisitos?**

   Planificar un proyecto de desarrollo del sistema de información.

   Entender el problema a resolver.

   Realizar un prototipo rápido.

3. **Según la norma IEEE 830, los requisitos deben cumplir con ciertos criterios de calidad como:**

   Correcto, su implementación es posible, evaluado, válido y actualizado, consistente.

Ambiguo, su implementación es posible, evaluado, válido y actualizado, consistente.

Correcto, su implementación es posible, evaluado, lenguaje altamente especializado, consistente.

4. **Los dos aspectos a ser considerados con mayor probabilidad en la elección de las técnicas de educción adecuadas para un sistema de gestión de datos son:**

   Los plazos del proyecto y el presupuesto, la edad de los implicados.

   Los plazos del proyecto y herramientas utilizadas, la disponibilidad de los implicados.

   Los plazos del proyecto y el presupuesto, la disponibilidad de los implicados.

5. **En qué fase del proceso de ingeniería de requisitos se cumplen las siguientes actividades: identificar a los involucrados, comprender el discurso del problema que resolverá el software, genera la lista de requisitos candidatos o preliminares.**

   Especificación de requisitos.

   Educción de requisitos.

   Validación de requisitos.

6. **En su proyecto se desarrolla un nuevo sistema de frenado para trenes de alta velocidad. Claramente, el resultado del desarrollo es un componente crítico para la seguridad del vehículo y debe cumplir varios requisitos de calidad. ¿Qué técnica de validación es adecuada en esta situación?**

   Prototipado.

   Inspección.

   Revisión guiada (“walkthrough”).

7. **A partir de los requisitos del sistema extraídos por el ingeniero de software se obtiene como producto final la especificación del sistema, que describe:**

   La planificación detallada de las etapas de una metodología a seguir en el proceso de construcción de software.

La lluvia de ideas que recoge los requisitos del usuario, producto de las reuniones preliminares.

Los modelos del sistema que obedecen a los requisitos funcionales y no funcionales de los usuarios.

8. **Entre las causas de los proyectos fallidos que se asocian con la ingeniería de requisitos son:**

   Falta de gestión de TI.

   Requisitos incompletos, falta de implicación de usuarios.

   Ausencia de apoyo de la Dirección.

9. **Las técnicas que se aplican para la extracción de requisitos son:**

   Entrevista, encuesta, lluvia de ideas.

   Pseudocódigo, lenguajes formales, observación.

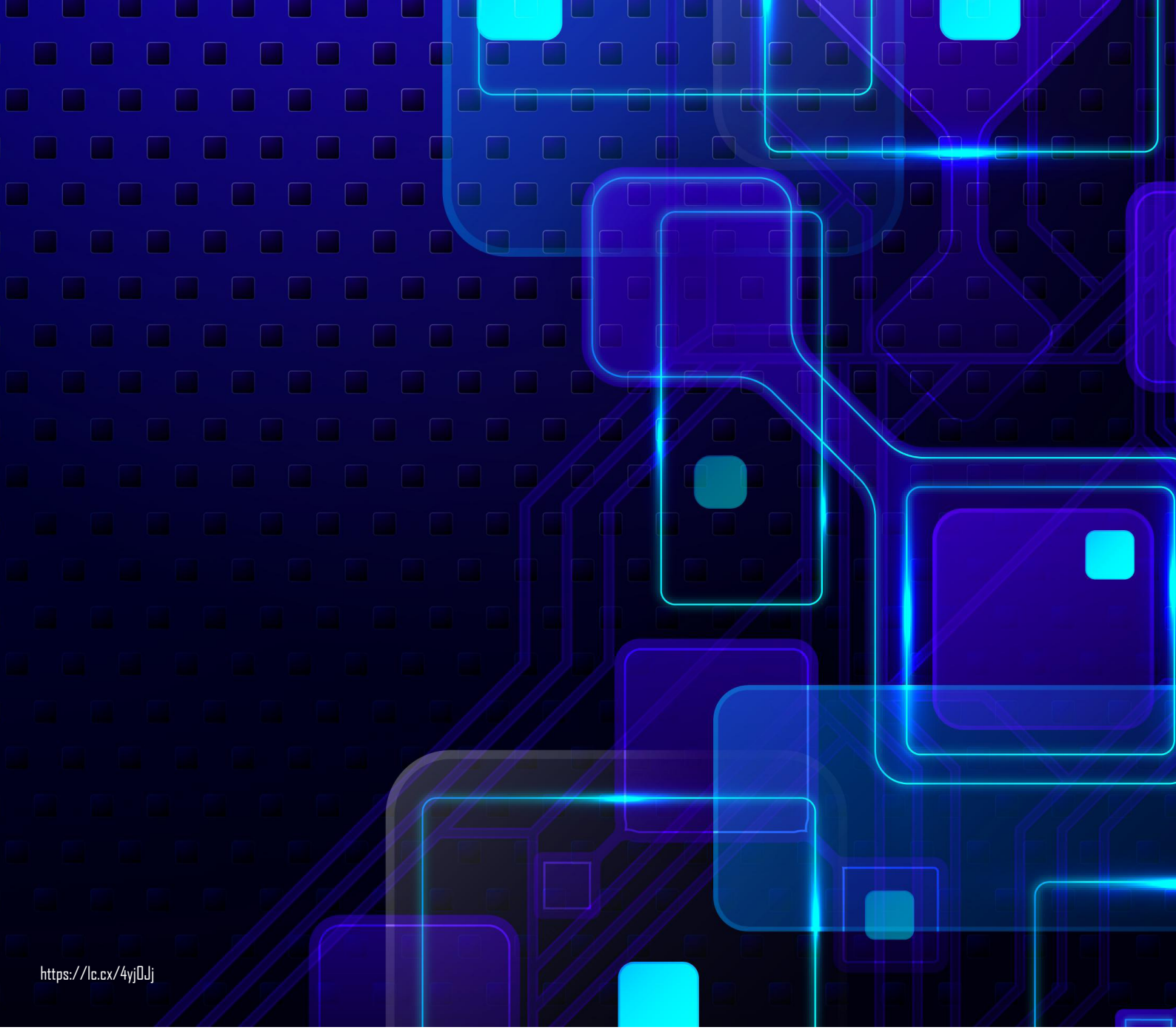
   Modelamiento conceptual, pseudocódigo, lluvia de ideas.

10. **Una característica fundamental que debe tener un ingeniero de requisitos es:**

    Alta creatividad para crear nuevos requisitos.

    Aptitud para investigar nuevas tecnologías.

    Capacidad para trabajar de manera colaborativa.



CAPÍTULO V

Análisis y diseño de software

Modelos conceptuales

El modelado de un producto software es el proceso usado para desarrollar modelos abstractos de un sistema, donde cada uno de estos modelos explica una perspectiva diferente de dicho producto, sin que por esto se pueda decir o pensar que son sistemas diferentes.

El proceso de modelado de sistemas, gracias a su practicidad, se ha convertido en un medio casi obligado para representar un sistema usando algún tipo de notación gráfica basado en notaciones, por ejemplo, el lenguaje de modelado unificado (UML). (Somerville, 2011)

Para Pressman 7 (2010) los modelos, en casi todas las disciplinas, se crean para entender mejor la realidad que se va a construir, transformar o desarrollar, el caso del software no es diferente, el modelo creado debe ser capaz de representar la información que el producto software transformará, la arquitectura y las funciones que permitirán que esto ocurra, las características que desean los usuarios y el comportamiento del producto software mientras la transformación tiene lugar (Tesuva, 2014).

La ingeniería de software crea dos clases de modelos: el de requisitos y el de diseño.

El primero, llamado también modelo de análisis, representa los requisitos establecidos por el cliente diferenciado en tres categorías diferentes: de información, de funcionalidad y de comportamiento. Por su parte, los modelos de diseño representan características propias del producto software, que permiten a los ingenieros desarrolladores de software elaborarlo de manera eficaz (Tesuva, 2014).

Modelos de análisis

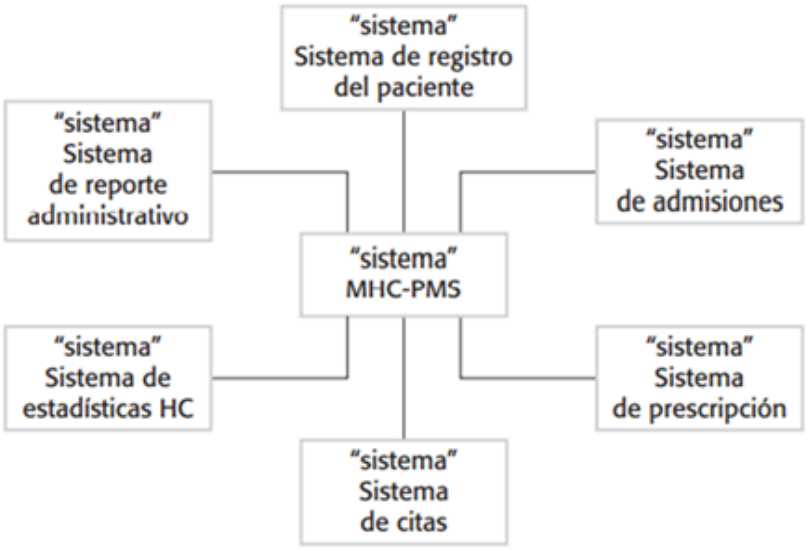
Los modelos de análisis son también conocidos como modelado de requisitos. Estos requisitos se los concibe como clases y se los extrae directamente del problema y su enunciado. Estas entidades o clases comúnmente representan datos o información que debe almacenarse en una base de datos y que persiste durante la vida útil del sistema.

En la primera etapa de especificación de requisitos de software (ERS) se considera la frontera del sistema, entendiéndose por esta a las características o

funcionalidades que se incluyen en el producto software y cuáles se ofertan en el entorno, como se muestra en la Figura 17 (Somerville, 2011).

Figura 17

El contexto de MHC-PMS

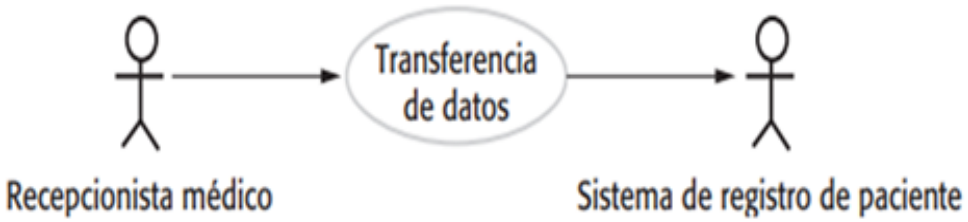


Nota. Procesos de ingeniería de requisitos. Adaptado de modelos de contexto https://modeladodelsistema.wordpress.com/2016/06/15/

Una de las aplicaciones más comunes de este tipo de modelamiento basado en el análisis, se la puede encontrar en los modelos de interacción, tales como, modelados de casos de uso, cuya función básica es establecer las interacciones entre un sistema y los actores externos que bien podrían ser usuarios propiamente dichos u otros sistemas. En la Figura 18 se muestra un diagrama de casos de uso y en la Figura 19 un diagrama de secuencia, ambos son modelos dinámicos que se emplean para modelar interacciones de componentes de un producto software.

Figura 18

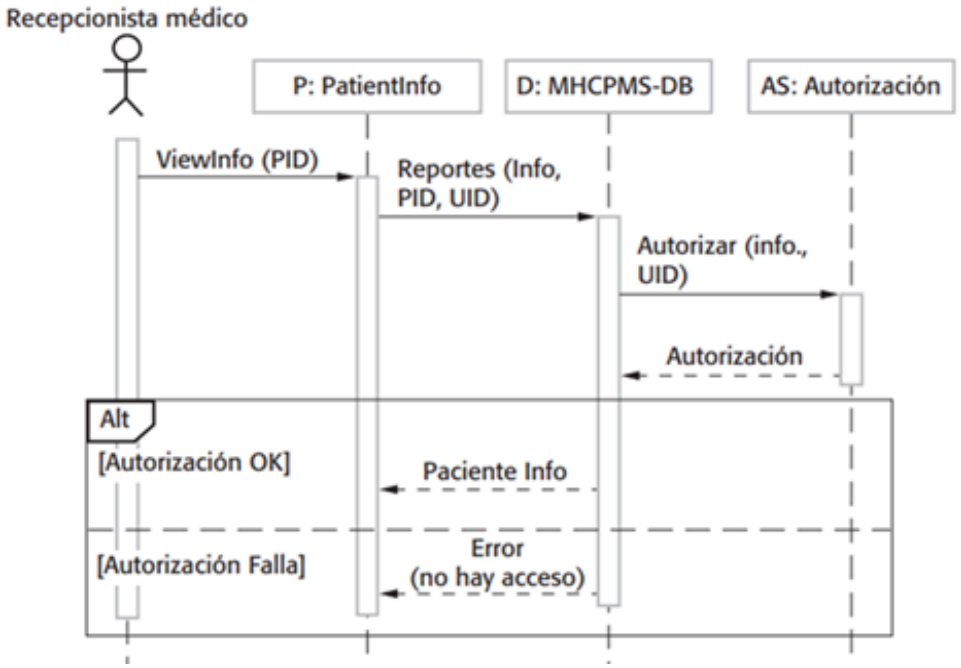
Caso de uso para la transferencia de datos



Nota. Modelos de interacción. Adaptado de casos de uso de transferencia de datos https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Figura 19

Diagrama de secuencia "Ver información del paciente"



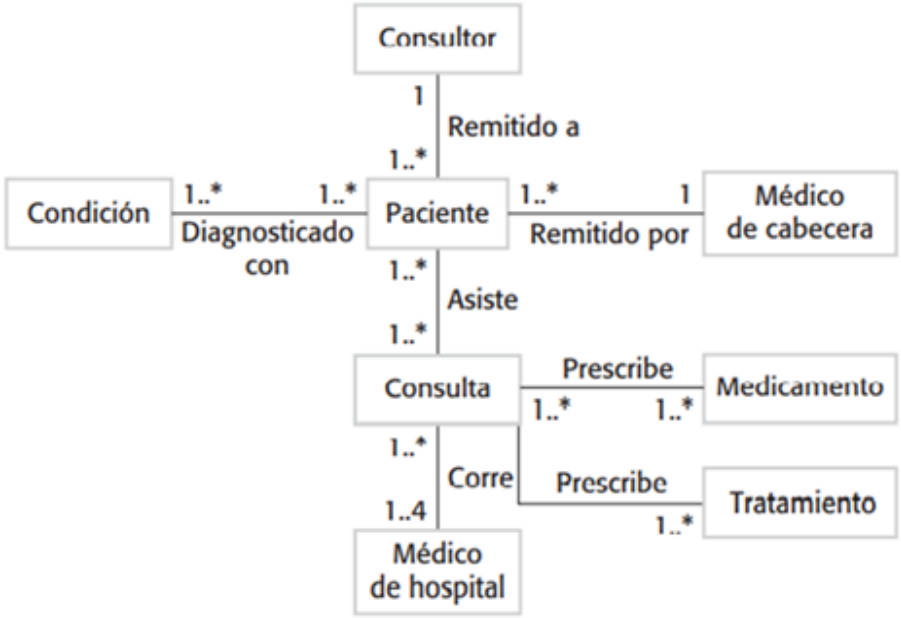
Nota. Modelos de interacción. Adaptado de diagramas de secuencia https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

En el caso de los modelos estructurales, que tienen por objetivo mostrar la organización de producto software, encontramos a los modelos estáticos y a los dinámicos; los modelos estáticos están diseñados para presentar la estructura del diseño del producto software, en tanto que los modelos dinámicos revelan la organización del producto software en el momento de su ejecución (Somerville, 2011).

Un ejemplo clásico de modelos estructurales son los diagramas de clases, utilizados cuando se desarrolla un modelo de producto software orientado a objetos, este modelo muestra las clases y sus asociaciones (Figura 20).

Figura 20

Clases y sus asociaciones en el MHC-PMS

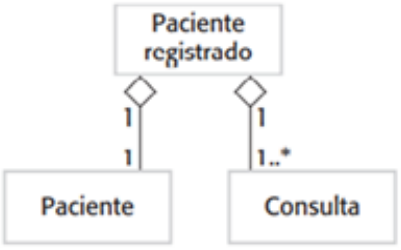


Nota. Modelos de interacción. Adaptado de diagramas de clases. https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Dentro de los modelos dinámicos que se usa en el desarrollo del producto software, encontramos también a los modelos de comportamiento, que toman ese nombre dado que, conforme se ejecuta el modelo, revelan la respuesta a un estímulo del entorno; estos modelos de comportamiento son de dos tipos: de datos y de eventos. El de datos toma información para ser procesada por el sistema; en tanto que el de los eventos son sucesos que activan el procesamiento del sistema. En la Figura 21 se muestra un ejemplo de agregación.

Figura 21

La asociación de la agregación

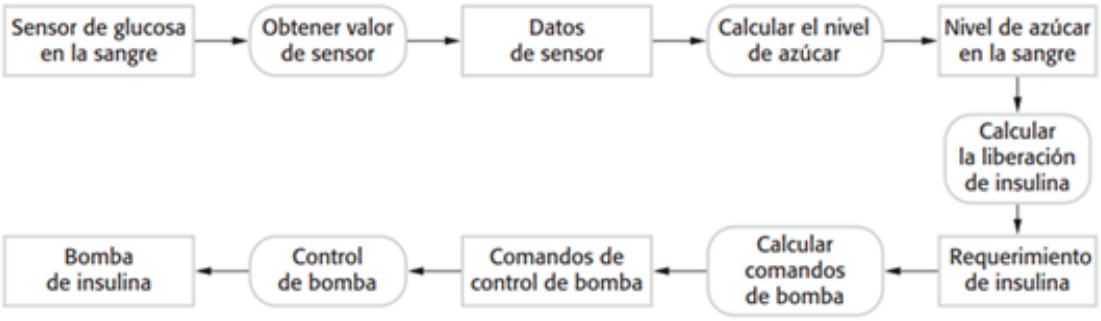


Nota. Modelos de comportamiento. Adaptado de diagramas de asociación agregación. Ejemplo de paciente registrado y la consulta asignada https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

En el modelado dirigido por datos se muestra la secuencia de acciones involucradas en el procesamiento de datos de entrada y la correspondiente generación de la salida asociada, por ejemplo, la Figura 22 presenta la cadena de procesamiento (simbolizados como actividades) y los datos que fluyen entre estos pasos (simbolizados como objetos).

Figura 22

Modelo de actividad de la operación de la bomba de insulina



Nota. Modelos dirigido por datos. Adaptado de modelo de actividad de operación de la bomba de insulina https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Modelos de diseño

Los modelos de diseño, llamados también modelos de sistema, presentan los objetos o clases de un sistema, así como las relaciones y las asociaciones entre tales entidades, de igual manera, se transforman en un puente entre los requisitos y el desarrollo del producto software incluida su implementación.

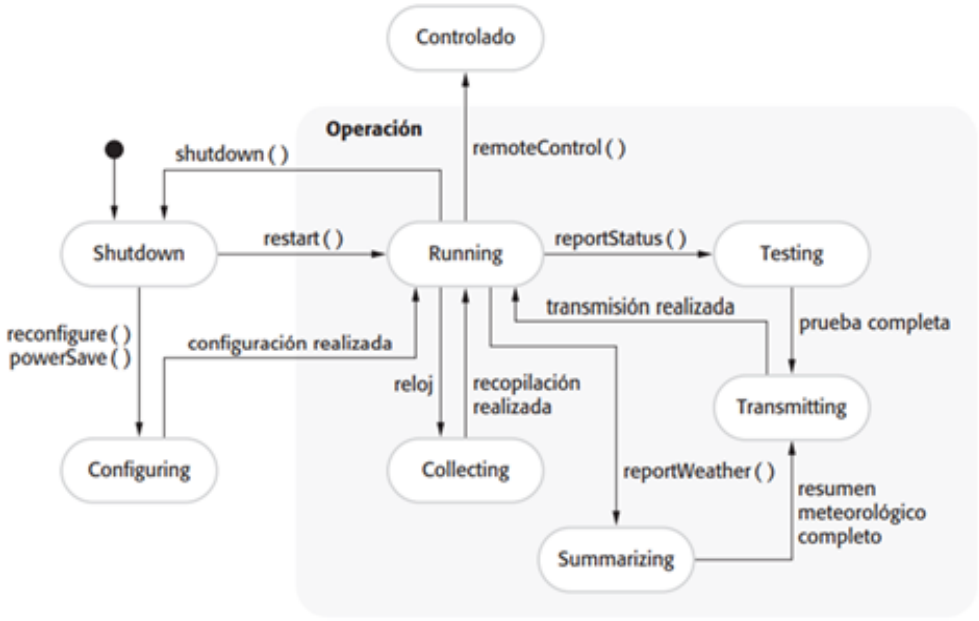
En el proceso de desarrollo de software es importante seleccionar qué modelos de diseño y a qué nivel de detalle se requieren. UML como herramienta de apoyo puede ayudar efectivamente en este proceso. En este caso, el uso de UML para elaborar modelos de diseño desarrollará modelos estructurales que describen la estructura estática del sistema, usando las clases de objetos y sus relaciones, y modelos dinámicos que explican la estructura dinámica y presentan las interacciones entre los objetos del sistema.

Al principio, en las primeras fases del proceso de diseño se recomienda utilizar solo tres modelos: Modelos de subsistema, que utilizan los diagramas de

clases (paquetes con objetos encerrados); Modelos de secuencia, que utilizan los diagramas de secuencia, y Modelos de máquina de estado, que presentan el cambio de estado de los objetos individuales en respuesta a ciertas condiciones o eventos. En la Figura 23, a manera de ejemplo, se indica el diagrama de estado para la estación meteorológica.

Figura 23

Diagrama de estado para la estación meteorológica



Nota. Modelos dirigido por datos. Adaptado de modelo de estación para la operación de la estación meteorológica https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Principios que guían el diseño de software

Los principios que guían el modelado de un producto software son:

1. El equipo de software tiene como objetivo principal, elaborar el software.
2. Viajar ligero, no crear más modelos de los necesarios.
3. Producir el modelo más sencillo que describa al problema o al software.
4. Construir modelos susceptibles al cambio.

5. Enunciar un propósito explícito para cada modelo que se cree
6. Adaptar los modelos que se desarrollan al sistema en cuestión.
7. Construir modelos útiles, olvidarse de elaborar modelos perfectos.
8. No ser dogmáticos respecto de la sintaxis del modelo.
9. Si su instinto dice que un modelo no es el correcto a pesar de que se vea bien en papel, hay razones para estar preocupado.
10. Obtener retroalimentación tan pronto como sea posible.

Estos principios son recomendaciones de Pressman 7 (Tesuva, 2014).

Recursos complementarios 5

Consulte el siguiente enlace para reforzar los conocimientos adquiridos:

Accede aquí:

Actividades de aprendizaje 5

Conteste las siguientes preguntas:

1. **La Ingeniería de software combina:**

   Diseño, codificación y pruebas de aceptación del usuario

   Codificación y pruebas

   Técnicas de ingeniería con prácticas de desarrollo de software

   Ninguna de las anteriores es cierta

2. **Las fases de proceso unificado de desarrollo PUD son:**

   Modelamiento del negocio análisis y diseño construcción pruebas y despliegue

   Comunicación planeación construcción pruebas implantación

   Incepción elaboración construcción transición

   Todas de las anteriores son ciertas

3. **¿Qué es un proceso de software?**

   Representa los roles de las personas involucradas en este proceso y las actividades de las que son responsables

   Representa el proceso como un conjunto de actividades cada una de las cuales realiza alguna transformación en los datos

   Conjunto de actividades cuya meta es el desarrollo u optimización de software

   Muestra la secuencia de actividades en el proceso junto con sus entradas salidas y dependencias

4. **Las notaciones que permiten representar la vista estática o la estructura del sistema son:**

   Diagramas de secuencia

   Diagramas de actividad

   Diagramas de casos de uso

   Diagramas de clases y objetos

5. **Se utilizan para representar modelos conceptuales de datos almacenados en repositorios de información esta información corresponde a:**

   Diagramas entidad relación

   Diagramas clases y objetos

   Diagramas de estados

   Diagramas de secuencia

6. **La aplicación de estándares de desarrollo externos o internos durante la construcción ayuda a lograr los objetivos de un proyecto, en lo referente a: eficiencia, calidad y costo. ¿Cuáles son los estándares de construcción que ayudan a lograr los objetivos de un proyecto?**

   Estándares para formatos y contenidos de documentos (nombres, diseño y sangría).

   Estándares para lenguajes como Java y C++ (lenguajes/ides de programación).

   Estándares de buenas prácticas de codificación (métodos de comunicación).

   Todas las anteriores son ciertas

7. **La ingeniería de software (SW) combina:**

   Diseño, codificación y pruebas de aceptación del usuario

   Técnicas de ingeniería con codificación. aceptación del usuario.

   Técnicas de ingeniería con prácticas de desarrollo de software.

   Codificación y pruebas

8. **Los equipos NO necesitan:**

   Trabajar juntos de forma colaborativa

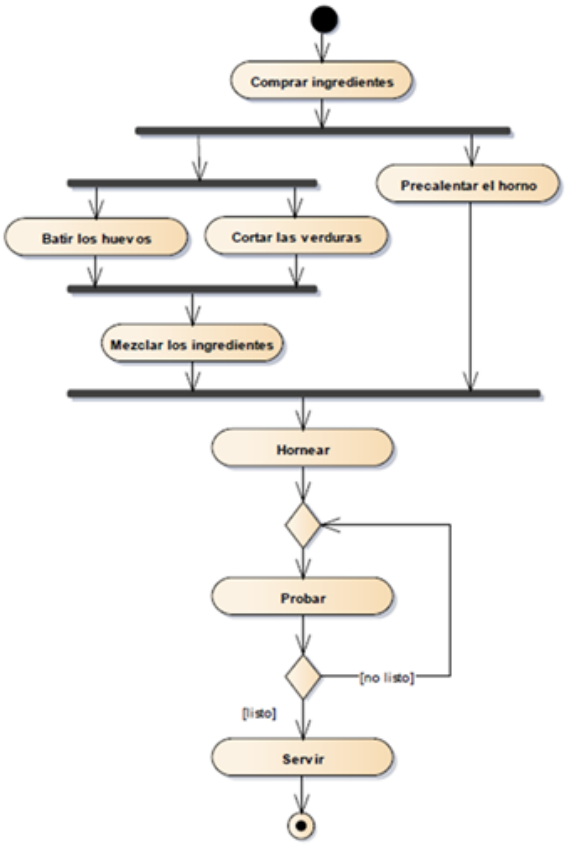
   Adaptarse continuamente al entorno cambiante

   Estar integrado por personal que posea las competencias y habilidades adecuadas

   Tener desarrolladores senior y jóvenes.

Autoevaluación del capítulo 5

1. **El siguiente diagrama de actividades muestra la preparación de una comida. ¿Cuál de las siguientes opciones mantiene la secuencia después de comprar los ingredientes?**



Batir los huevos y cortar las verduras son acciones previas antes de pasar a mezclar los ingredientes.

Mezclar los ingredientes ocurre tan pronto como cortar las verduras.

Probar que tiene que seguir directamente a servir.

2. **Cómo se construye un buen sistema de información (SI) considerando que el punto de partida es:**

   Utilizar herramientas de desarrollo como medio para alcanzar un producto de calidad.

   Utilizar un proceso definido con fases claras, donde cada una de estas genera un producto final.

   Las pruebas y validaciones no son indispensables para la construcción del producto.

3. Según las siguientes definiciones, el orden correcto de las palabras es:

   .....es un conjunto de técnicas y de procedimientos organizados en fases para el desarrollo de productos software, de manera eficaz y abarca el ciclo de vida del mismo.

   .....es una técnica repetible para la resolución de un problema específico.

   .....…….. es un conjunto de reglas gráficas o textuales para representar un modelo.

   Metodología/Método/Notación.

   Metodología/Notación/Método.

   Método/Metodología/Notación.

4. **Las notaciones que permiten representar la vista estática o la estructura del sistema son:**

   Diagramas de secuencia.

   Diagramas de actividad.

   Diagramas de clase y objeto.

5. **Se utilizan para representar modelos conceptuales de datos almacenados en repositorios de información. Esta definición corresponde a:**

   CRC.

Diagramas entidad relación.

Diagramas de implementación.

6. **Las notaciones que permiten representar la vista dinámica o comportamiento del sistema son:**

   Diagramas de componentes.

   Diagramas de transición de estados.

   Diagramas de comunicación.

7. **¿Qué es refinamiento?**

   Es una medida de la fuerza de asociación de los elementos dentro de un módulo, cohesión.

   Es una vista de un objeto que se enfoca en la información relevante para un propósito particular e ignora el resto de la información, abstracción.

   Consiste en seguir una estrategia descendente a la hora de diseñar.

8. **En qué fase del proceso de ingeniería de requisitos se cumplen las actividades de identificar a los involucrados, comprender el discurso del problema que resolverá el software, generar la lista de requisitos candidatos o preliminares.**

   Especificación.

   Elicitación.

   Análisis.

9. **Una característica del software es:**

   El software usa componentes estándar con funciones de interfaces bien definidas.

   El software se desgasta con el transcurso del tiempo.

   El software es un elemento lógico.

10. **Los elementos que componen el software son:**

    Programas, procedimientos, documentación y datos relacionados.

    Personal, proceso y producto.

    Programas o instrucciones, partes y piezas y datos

CAPÍTULO VI

Construcción de software

Transformación de modelos en código

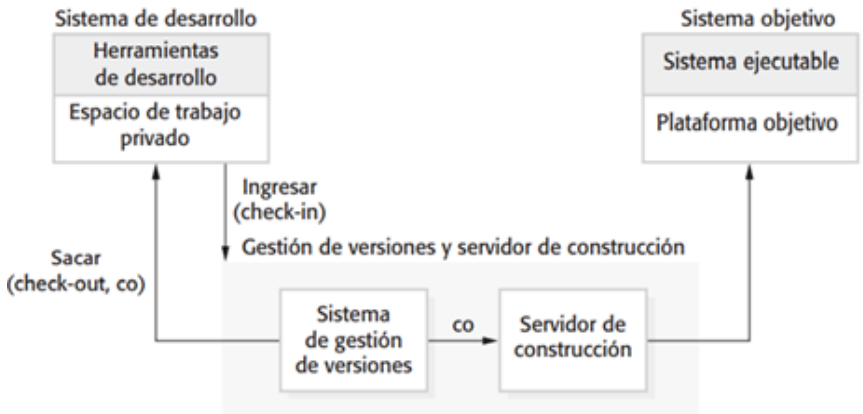
La construcción de un producto software es un proceso estandarizado y metódico que permite compilar los componentes o las unidades que lo forman, vinculándolos con otros elementos a fin de crear un programa ejecutable. Generalmente, este proceso es automatizado y se pueden usar varias herramientas, una de ellas es el lenguaje Java (Somerville, 2011).

En la Figura 24 se indica cómo la plataforma de desarrollo, el proceso de construcción y el objetivo interactúan con el fin de generar un producto software ejecutable. En el Sistema de desarrollo se elaboran los códigos mediante una diversidad de herramientas entre las que se pueden nombrar a los editores, IDES de desarrollo, compiladores, etc.

Los desarrolladores, por su parte, remiten su código a un espacio de trabajo privado donde se generan las diferentes versiones que son enviadas al Sistema de gestión de versiones. El servidor de construcción se encarga de desarrollar versiones ejecutables definitivas del producto software. El entorno objetivo, en cambio, es la plataforma donde se ejecuta el sistema y depende del tipo de producto a desarrollar, que puede ser un entorno móvil o web; por otro lado, es necesario recordar que no es posible realizar las actividades de construcción y prueba al mismo tiempo en este servidor, habida cuenta que la construcción y las pruebas son etapas diferentes del proceso, y como tal están ejecutadas por diferentes equipos de personas en distintas etapas del mismo.

Figura 24

Plataforma de desarrollo, construcción y objetivo



Nota. Administración de la configuración. Adaptado de construcción del sistema https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

El sistema de construcción de un producto software requiere de la concurrencia e interacción de gran cantidad de información, en la Figura 25 se muestra que el proceso constructivo requiere de un IDE, archivos, librerías, versiones del compilador, y resultados de pruebas entre otras.

Figura 25

Construcción del sistema

Nota. Adaptado de construcción del sistema https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Estándares para la construcción

Los estándares de construcción, llamados también de desarrollo, que pueden ser externos e internos, permiten alcanzar el objetivo de un proyecto en lo que a costo, calidad y eficiencia se refieren. Los que inciden directamente en los problemas de construcción son los siguientes:

Métodos de comunicación, por ejemplo, estándares de formatos y contenidos, y lenguajes de programación

Lenguajes de programación como son Java. C++, C#, Python

Estándares de codificación para convenciones de nombres, diseño y sangría. También podría darse el caso de uso de estándares internos, que normalmente obedecen a requerimientos corporativos o especiales creados para usos en proyectos tipo.

El cuerpo de conocimiento de la ingeniería de software (IEEE Computer, 2014) como se muestra en la Figura 26, el proceso completo para la construcción de software, en este se incluyen los siguientes principios fundamentales:

minimizar la complejidad, anticipar los cambios, construir para verificar y reutilizar (IEEE Computer, 2014).

Minimizar la complejidad aun minimizando el cambio, la capacidad de la mayoría de las personas es limitada para tener en mente estructuras complejas de información, en periodos largos de tiempo, de ahí la necesidad de reducir la complejidad aplicada al proceso de construcción (IEEE Computer, 2014).

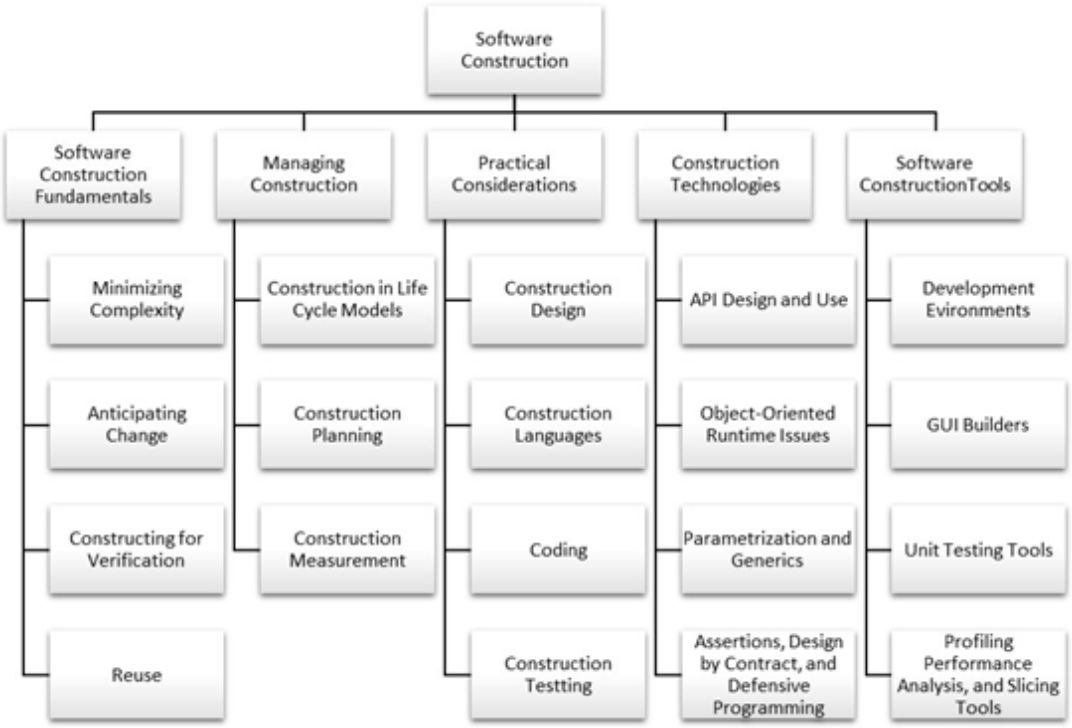
Anticipar cambios dado que la construcción del producto software cambiará con el tiempo, anticiparse al cambio ayudará a crear software extensible, es decir, que este producto puede mejorar sin cambiar su estructura imperfecta (IEEE Computer, 2014).

Construir para verificar representa construir productos software de tal forma que quienes escriben software, los probadores o tester y los usuarios en general puedan examinar, detectar las fallas fácilmente (IEEE Computer, 2014).

Reutilizar es el uso en nuevas aplicaciones de activos preexistentes que han sido probados satisfactoriamente y que permitan resolver diversos problemas, por ejemplo, reutilización de bibliotecas, módulos, componentes, código fuente, activos comerciales, entre otros (IEEE Computer, 2014).

Figura 26

Desglose de temas para la construcción de software



Nota. Construcción de software. Adaptado de tópicos de la construcción del cuerpo de conocimiento de la Ingeniería de Software. https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf

Principios guía para la construcción

Los principios que guían la construcción de un producto software están íntimamente relacionados con el estilo de programación, a saber:

Planear la codificación antes de iniciar la codificación, asegúrese de entender el problema a resolver, comprender los conceptos y principios de diseño, seleccionar un IDE de desarrollo que satisfaga las necesidades del producto software a construir y la operatividad de su entorno, y desarrollar las pruebas unitarias que serán aplicadas una vez terminada la codificación de este (Tesuva, 2014).

Codificar al escribir códigos, asegúrese de limitar los algoritmos de acuerdo con el paradigma de programación estructurada, considere también usar programación por pares; al seleccionar las estructuras de datos, tome en cuenta aquellas que cumplan con el diseño y procure usar una lógica condicional simple. Al nombrar variables que sean significativas, use nombres que no causen confusión y cree un diseño visual utilizando recursos como sangrías o líneas en blanco, que permitan la comprensión del código sin dificultad (Tesuva, 2014).

Validar una vez terminada la primera fase de codificación, realice las pruebas unitarias necesarias que permitan verificar la funcionalidad de los requisitos y corregir los errores descubiertos al momento (Tesuva, 2014).

Prueba es un proceso de ejecución de un programa con la intención de encontrar un error, la prueba exitosa es aquella que descubre un error oculto (Tesuva, 2014).

Recursos complementarios 6

Consulte el siguiente enlace para reforzar los conocimientos adquiridos:

Accede aquí:

Actividades de aprendizaje 6

1. **Empareje la información del siguiente cuadro:**

Programación	Ing. de Software
a. Solo un desarrollador	1. Equipos de desarrollo
b. Aplicaciones de juguete	2. Sistemas roles
c. Uno o pocos involucrados	3. Sistemas complejos
d. Mantenimiento mínimo	4. Mantenimiento es + 60% costos de desarrollo

Opc. A a1, b2, c3, d4 Respuesta 1 OK ( ) NOK ( )

Opc. B. a4, b2, c3, d1 Respuesta 2 OK ( ) NOK ( )

Opc D. a3, b2, c1, d4 Respuesta 3 OK ( ) NOK ( )

Opc. C. a2, b1, c3, d4 Respuesta 4 OK ( ) NOK ( )

2. **Seleccione las definiciones de diagramas de UML, con el término correspondiente.**

1. Muestran el flujo de control de una actividad. Se puede usar para representar actividades concurrentes.	A. Diagramas de actividad/ Secuencia/ Transición de estados/ Comunicación.
2. Muestran las interacciones entre un grupo de objetos, con énfasis en el orden de tiempo de los mensajes que se pasan entre los objetos.	B. Diagramas de actividad/ Secuencia/ Transición de estados/ Comunicación.
3. Muestran el flujo de control de estado a estado y cómo el comportamiento de un componente cambia en función de su estado actual en una máquina de estados	C. Diagramas de actividad/ Secuencia/ Transición de estados/ Comunicación.
4. Muestran las interacciones que ocurren entre un grupo de objetos; se hace hincapié en los objetos, sus enlaces, y los mensajes que intercambian en estos enlaces	D. Diagramas de actividad/ Secuencia/ Transición de estados/ Comunicación.

3. **¿Qué es un proceso de software? Seleccione una respuesta**

   Muestra la secuencia de actividades en el proceso junto con sus entradas, salidas y dependencias, las actividades en este modelo representan acciones humanas.

   Es un conjunto de actividades cuya meta es el desarrollo u optimización del software.

   Representa el proceso como un conjunto de actividades, cada una de las cuales realiza alguna transformación en los datos.

   Representa los roles de las personas involucradas en el proceso de software y las actividades de las que son responsables.

4. **Cómo se construye un buen sistema de información (SI) considerando que el punto de partida es:**

   Utilizar herramientas de desarrollo como medio para alcanzar un producto de calidad

   La definición de requisitos claros es una parte del proceso, pero no es relevante

   Las pruebas y validaciones no son indispensables para la construcción del producto

   Utilizar un proceso definido con fases claras, donde cada una de estas genera un producto final.

5. **¿Qué es una metodología de desarrollo de software?**

   Un conjunto de métodos que cubren todo el ciclo de vida de desarrollo de sistemas, y que están unidos por un enfoque general o filosófico.

   Es un conjunto de lenguajes de programación que permiten analizar, diseñar y construir productos software.

   Es una herramienta para resolver problemas determinísticos.

   Un conjunto de rutinas de programación que permiten desarrollar aplicaciones de forma ágil.

6. **Ingeniería de software asistida por computador significa:**

   Conjunto de herramientas de software integradas para ser utilizadas por otras

   Elementos físicos del computador integrados para ser utilizados por otras.

Conjunto de herramientas de hardware integradas para ser utilizadas por otras.

Todas las anteriores.

7. **Identifique las notaciones que permiten representar la vista dinámica o comportamiento del sistema:**

   Diagramas de clase y objeto

   Diagramas de transición estados

   Diagramas de implementación

   Diagramas de comunicación

   Diagramas de componentes

   Diagramas de secuencia

   Tarjetas de clase-responsabilidad-colaboración (CRC)

   Diagramas de actividad

8. **Respecto de la reparación de fallas, indique en qué fase es más costoso realizar las reparaciones o correcciones, en caso de detectar errores:**

   En la fase de programación

   En la fase de operación del sistema

   En la fase de ingeniería de requisitos

   En la fase de diseño

Autoevaluación del capítulo 6

1. **La aplicación de estándares de desarrollo externos o internos durante la construcción ayuda a lograr los objetivos de un proyecto, en lo referente a: eficiencia, calidad y costo. ¿Cuáles son los estándares de construcción que ayudan a lograr los objetivos de un proyecto?**

   Estándares para formatos y contenidos de documentos (nombres, diseño y sangría).

   Estándares para lenguajes como Java y C++ (lenguajes/ides de programación).

   Estándares de buenas prácticas de codificación (métodos de comunicación).

2. **La construcción tiene como meta aplicar los atributos de un buen software, ¿cuáles son estos atributos?**

   Al gastar los recursos del sistema.

   Fácil de utilizar, con esfuerzo adicional por parte del usuario.

   Mantenible, confiable y fácil de usar.

3. **Usted es el líder del equipo de desarrollo de un aplicativo que requiere maximizar la precisión, el comportamiento del tiempo y la capacidad de prueba; así también debe minimizar la ambigüedad. Por lo tanto, usted decidirá un lenguaje de programación que se base en qué tipo de notaciones.**

   Notaciones visuales.

   Notaciones lingüísticas.

   Notaciones formales.

4. **Usted construye software de tal manera que los ingenieros de software, así como los tester (probadores) y los usuarios puedan encontrar fácilmente las fallas durante las pruebas y actividades operativas independientes. ¿Qué concepto clave de la construcción de software aplicaría?**

   Construyendo para la verificación.

   Minimizando la complejidad.

   Anticipando el cambio.

5. **Las actividades estructurales del proceso de desarrollo de software, consideradas en la construcción de software, son:**

   Comunicación, despliegue, modelado y administración del proyecto.

   Comunicación, despliegue, modelado y diseño.

   Comunicación, despliegue, modelado y despliegue.

6. **Las actividades de protección o sombrilla del proceso de desarrollo de software, consideradas en la construcción de software, son:**

   Gestión de la configuración, comunicación, revisiones técnicas, garantía de calidad.

   Gestión de la configuración, gestión de riesgos, revisiones técnicas, garantía de calidad.

Gestión de la configuración, comunicación, revisiones técnicas, programación.

7. **El tipo de software de ingeniería y científico es:**

   Software que se caracteriza por el uso de algoritmos de manejo de números. Las aplicaciones van desde la astronomía a la vulcanología, dinámica orbital de las lanzaderas espaciales, biología molecular, entre otras.

   Software que reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo.

   Software que hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo.

8. **De las siguientes opciones, cuál es la correcta en relación con la programación versus la ingeniería de software.**

   Solo un desarrollador (equipos de desarrollo), aplicaciones de juguete (sistemas, roles). Uno o pocos involucrados (mantenimiento es +60 % de costos de desarrollo), mantenimiento mínimo (sistemas complejos).

   Solo un desarrollador (equipos de desarrollo), aplicaciones de juguete (sistemas, roles). Uno o pocos involucrados (sistemas complejos), mantenimiento mínimo (mantenimiento es +60 % de costos de desarrollo).

   Solo un desarrollador (sistemas, roles), aplicaciones de juguete (equipos de desarrollo). Uno o pocos involucrados (sistemas complejos), mantenimiento mínimo (mantenimiento es +60 % de costos de desarrollo).

9. **El responsable de la gestión de la configuración de software que garantiza la coherencia y la integridad del código es:**

   El auditor.

   El administrador de configuración.

   El desarrollador.

10. **Las causas de proyectos fallidos que están asociadas a la ingeniería de requisitos en la construcción de un producto software son:**

    Requisitos funcionales incompletos, falta de compromiso de los usuarios.

    Requisitos funcionales incompletos, falta de gestión de TI.

    Requisitos funcionales incompletos, falta de apoyo de la dirección de TI.

CAPÍTULO VII

Prueba de software

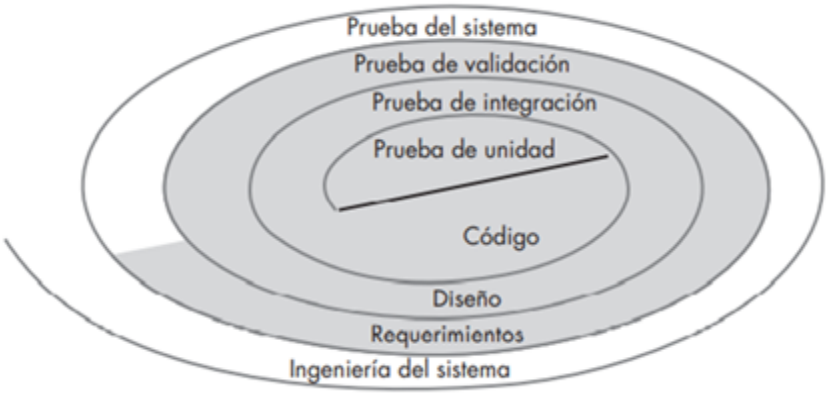
Estrategias de prueba

Las estrategias de prueba son un conjunto de buenas prácticas de ingeniería de software que permiten planificar una serie de pasos necesarios para aplicar diferentes casos de prueba. Según el momento y la necesidad son pertinentes diferentes estrategias de pruebas, estas son ejecutadas por personas del equipo de desarrollo dedicadas a este fin, con el único objetivo de validar el cumplimiento de los requisitos funcionales solicitados por el usuario.

En la Figura 27 se presenta una espiral en sentido horario de las manecillas del reloj, con esta espiral se busca representar un esquema de lo que se debe entender por estrategias de pruebas. Como se mencionó anteriormente, para cada etapa del proyecto son pertinentes y necesarias diferentes tipos de pruebas. Las pruebas inician en el núcleo o matriz de la espiral, donde se congregan a componentes, clases y a los objetos de contenido implementados en el código fuente. A medida que esta avanza hacia afuera, en el primer paso de esta espiral encontramos a la prueba de integración, centrada en el diseño y la construcción de la arquitectura de software. En el segundo paso tenemos a la prueba de validación, donde se valida que el producto software cumpla con los requisitos establecidos por el usuario. Y en el tercer paso se encuentra la prueba de sistema, donde los elementos del sistema y el software se prueban como un todo (Tesuva, 2014).

Figura 27

Estrategia de pruebas



Nota. Pruebas de software. Adaptado de estrategias de pruebas https://www.researchgate.net/figure/Figura-1-Estrategias-de-Prueba-Pressman-2001_fig1_318457069

Técnicas de pruebas

El proceso de pruebas como tal tiene dos objetivos básicos: explicar tanto al desarrollador como al usuario que el producto software cumple con los requisitos y localizar situaciones donde el comportamiento del producto software no esté de acuerdo con los requisitos establecidos.

El primero corresponde a la prueba de validación, donde los conjuntos de casos de prueba buscan un cumplimiento correcto de los requisitos establecidos por el usuario. Y el segundo, está orientado a encontrar defectos o bugs del producto software en determinadas condiciones. En la Figura 28, Boehm, quien dio los primeros inicios de la Ingeniería de software, expresó brevemente la diferencia entre validación y verificación, con dos simples preguntas:

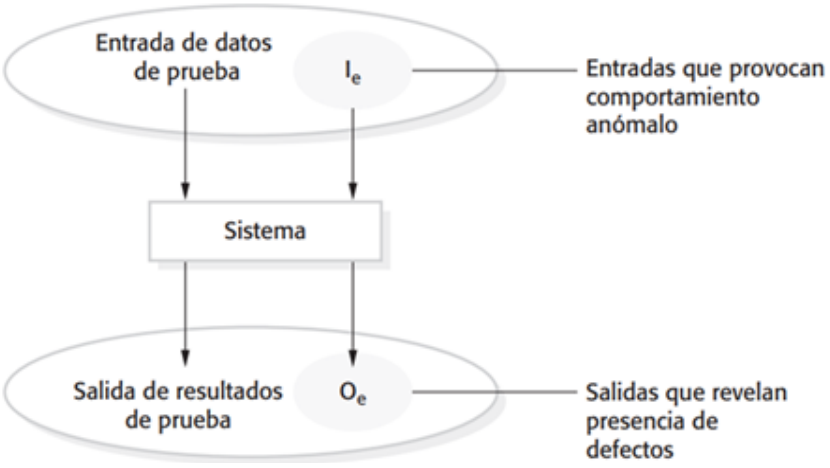
"Validación: ¿Construimos el producto correcto?"

"Verificación: ¿Construimos bien el producto?". (García, 2018)

Los dos procesos tienen como objetivo buscar y comprobar que el producto software en construcción respete sus especificaciones y manifieste la funcionalidad deseada por los usuarios".

Figura 28

Modelo de entrada y salida de una prueba de programa



Nota. Pruebas de software. Adaptado de pruebas de desarrollo https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Casos de prueba

Los casos de prueba no son más que las especificaciones de las entradas de la prueba, y la salida esperada del sistema o resultados de prueba, además de la información detallada del componente o sujeto que se está probando (Somerville, 2011).

De forma general, un producto software debe pasar por varios tipos de pruebas que indistintamente de su nombre o método de aplicación se las puede agrupar en tres tipos:

1. De desarrollo, donde el producto software se pone a prueba durante todas las etapas del proceso de desarrollo a fin de descubrir bugs (errores) y defectos.
2. Versiones de pruebas donde, a través de otro equipo, se prueba una versión completa del producto software.
3. De usuario, donde usuarios reales prueban el producto en su propio entorno (Somerville, 2011).

Las pruebas de desarrollo contienen todas las actividades de prueba que realizan los desarrolladores para validar y verificar el producto software; en esta etapa el rol de Tester o probador le corresponde al programador que diseñó el producto software. Estas pruebas de desarrollo se efectúan en tres distintos niveles:

1. Pruebas de unidad, donde los programas o clases de objetos, ejecutan individualmente una prueba orientada a comprobar la funcionalidad de los objetos de estos.
2. Pruebas de componentes, que toman las interfaces desarrolladas a fin de probar su funcionalidad.
3. Pruebas de sistema, en las que se prueban las interacciones de los componentes. Cada uno de estos tipos de pruebas son procesos de verificación de defectos donde el objetivo es descubrir bugs (Somerville, 2011).

Las técnicas de pruebas que pueden ser aplicadas a un producto software son: de caja blanca y de caja negra.

Las pruebas de caja blanca, llamada también pruebas estructurales, son un enfoque sistémico a las pruebas donde se usa el conocimiento del código fuente de programa para diseñar pruebas de defecto.

Las pruebas de caja negra, llamadas también pruebas funcionales, son derivadas a partir de la especificación del producto software, este es tratado como una caja negra cuyo comportamiento sólo puede determinarse por el estudio de entradas y salidas relacionada (Somerville, 2011).

A continuación, se mencionará al menos una técnica para prueba de caja blanca, como es la prueba de ruta básica y una para caja negra, como es la prueba de partición equivalente.

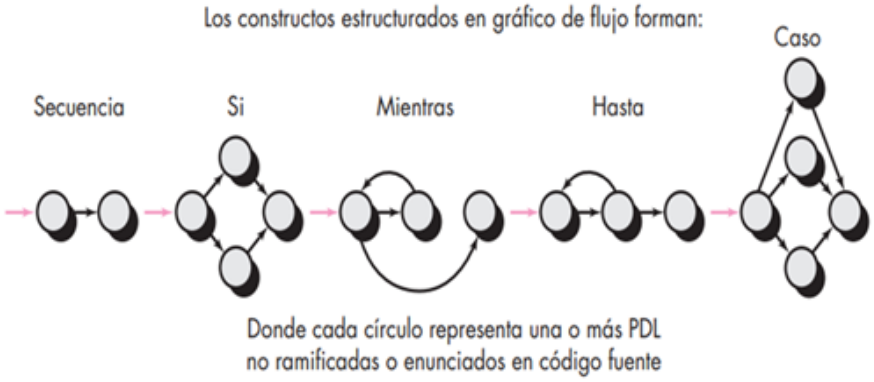
La prueba de ruta básica es una técnica de caja blanca propuesta por Tom McCabe, que permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño de procedimiento y usar esta medida para definir cuántos caminos podrían resultar de esta prueba (Tesuva, 2014).

El proceso para la ruta básica presenta los siguientes pasos:

1. Seleccione un pedazo de código fuente que incluya estructuras condicionales, repetitivas, como, por ejemplo, If-then-else, repeat, until, swich–case, numere cada estructura y transforme este en un diagrama de flujo de datos. Transforme el diagrama de flujo de datos en Grafo de flujo según se indica en la Figura 29 Notación del grafo de flujo. También puede observar este paso en la Figura 30 Diagrama de flujo de datos y grafo de flujo.
2. Calcule la complejidad ciclomática V(G) =E-N+2 donde E=número de aristas del grafo de flujo (flechas que entran o salen de cada nodo) N=-número de nodos del grafo de flujo, la V(G)= P+1 donde P= número de nodos predicado o estructuras condicionales (IF-Then-Else).
3. Determine el conjunto de caminos básicos resultantes del cálculo de V(G), (Tesuva, 2014).

Figura 29

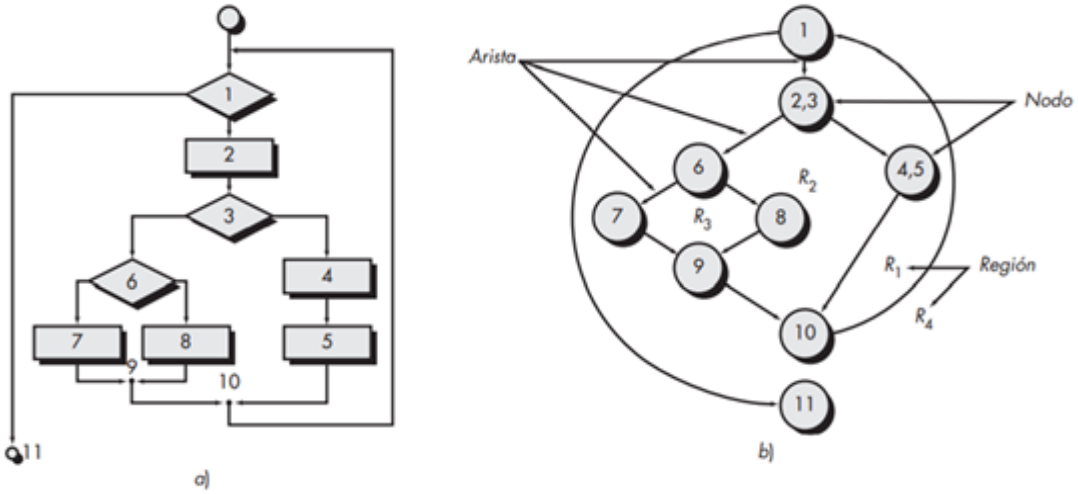
Notación del grafo de flujo



Nota. Técnicas de pruebas de software. Adaptado de notación de grafos de flujos Pressman, https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

Figura 30

Diagrama de flujo de datos y grafo de flujo



Nota. Técnicas de pruebas de software. Adaptado de notación de grafos de flujos Pressman, https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

La partición de equivalencia es un método de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos, de los cuales pueden derivarse casos de prueba. Para la práctica correspondiente, se suministrará una tabla donde podrá preparar estos casos. Considere las siguientes indicaciones:

1. Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos inválidas.
3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida
4. Si una condición de entrada es booleana, se define una clase de equivalencia válida y una inválida.

En la Tabla 1 se presenta un ejemplo de clase de equivalencia producto de un trabajo académico.

Tabla 1

Partición de equivalencias ejemplo

Variable	Clase de equivalencia	Estado	Representante
Nombres-Apellidos	EC1: Nombre-Apellido == TextNombre,TextApellido	Válido	Luis Tavaris
	EC2: Nombre-Apellido!= TextNombre, TextApellido	Válido	----------

Nota. Partición de equivalencias. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/ja-ruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

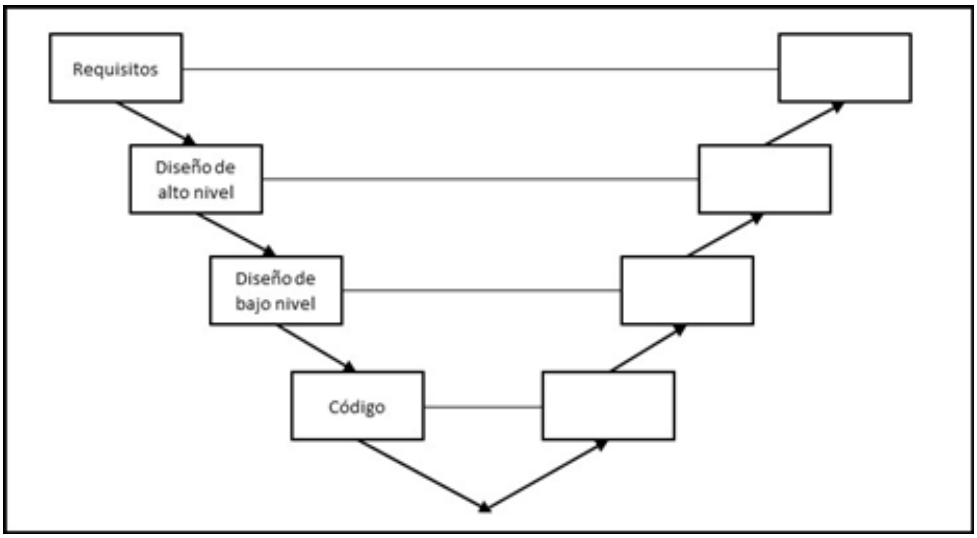
Recursos complementarios 7

Consulte los siguientes enlaces para reforzar los conocimientos adquiridos:

Accede aquí:

Autoevaluación del capítulo 7

1. **Las pruebas de software generalmente se realizan en diferentes niveles a lo largo de los procesos de desarrollo y mantenimiento; los niveles se pueden distinguir en función del objeto de prueba. En el siguiente gráfico del modelo V, indique el correspondiente nivel de prueba.**



Pruebas unitarias, pruebas de integración, pruebas caja negra, pruebas caja blanca

Pruebas de aceptación, pruebas de sistema, pruebas de integración, pruebas unitarias

Pruebas de sistema, pruebas de aceptación, pruebas de integración, pruebas unitarias

2. **Las pruebas que corresponden a verdaderos principios de prueba son:**

Las pruebas tempranas ahorran tiempo y dinero.

Las pruebas no dependen del contexto.

Las pruebas exhaustivas son posibles.

3. **¿En qué fase es más costoso realizar las reparaciones o correcciones en caso de detectar errores en la reparación de fallas?**

En la fase de ingeniería de requisitos.

En la fase de ingeniería de programación.

En la fase de operación del producto software (sistema).

4. **Son las pruebas que comprueban la lógica interna del código fuente, centrándose en el comportamiento y la estructura interna del código. Nos referimos a:**

   Pruebas de sistema.

   Pruebas unitarias.

   Pruebas de caja blanca.

5. **Son las pruebas que se centran únicamente en los datos de entrada y los datos de salida, sin tomar en cuenta los detalles internos de código fuente. Se trata de:**

   Pruebas de integración

   Pruebas de caja negra

   Pruebas unitarias

6. **El Sr. Probador ha probado aplicaciones de productos software en dispositivos móviles durante cinco años, tiene mucha experiencia en pruebas de aplicaciones móviles y logra mejores resultados en menos tiempo que otros. Durante varios meses no modificó los casos de prueba automatizados existentes, tampoco creó casos de prueba nuevos, por esta razón no encuentra defectos al ejecutar las pruebas. ¿Qué principio de prueba no fue observado por parte del Sr. Probador?**

   La repetición de los mismos casos de prueba no permitirá encontrar nuevos defectos.

   Los defectos se agrupan juntos.

   Las pruebas dependen del entorno.

7. **¿Cuál es un objetivo válido para la prueba?**

   Comenzar lo más tarde posible para que el desarrollo tenga tiempo suficiente para crear un buen producto.

   Probar que cualquier defecto restante no causará fallas.

   Validar si el objeto de prueba funciona según lo esperado por los usuarios y otras partes interesadas.

8. **Pressman menciona que existe un tipo de prueba para cada fase del proceso de desarrollo de un producto software. De las siguientes opciones, cuál es la correcta.**

Prueba de unidad-ingeniería del sistema, prueba de integración-diseño, prueba de validación-requisitos, prueba del sistema-código

Prueba de unidad-código, prueba de integración-diseño, prueba de validación-requisitos, prueba del sistema-ingeniería del sistema para validar si el objeto de prueba funciona según lo esperado por los usuarios y otras partes interesadas.

Prueba de unidad-diseño, prueba de integración-código, prueba de validación-requisitos, prueba del sistema-ingeniería del sistema

9. **La prueba de la ruta básica tiene varios pasos secuenciales. El orden correcto es:**

   Transforme en diagrama de flujo de datos, este pase al grafo de flujo, calcule la complejidad ciclomática, determine el número de rutas a ser recorridas al menos una vez.

   Seleccione código fuente, transforme en diagrama de flujo de datos, este diagrama de flujo transforme al grafo de flujo, calcule la complejidad ciclomática, determine el número de rutas a ser recorridas al menos una vez.

   Seleccione código fuente, transforme en diagrama de flujo de datos, este pase al grafo de flujo, calcule la complejidad ciclomática, determine el número de rutas.

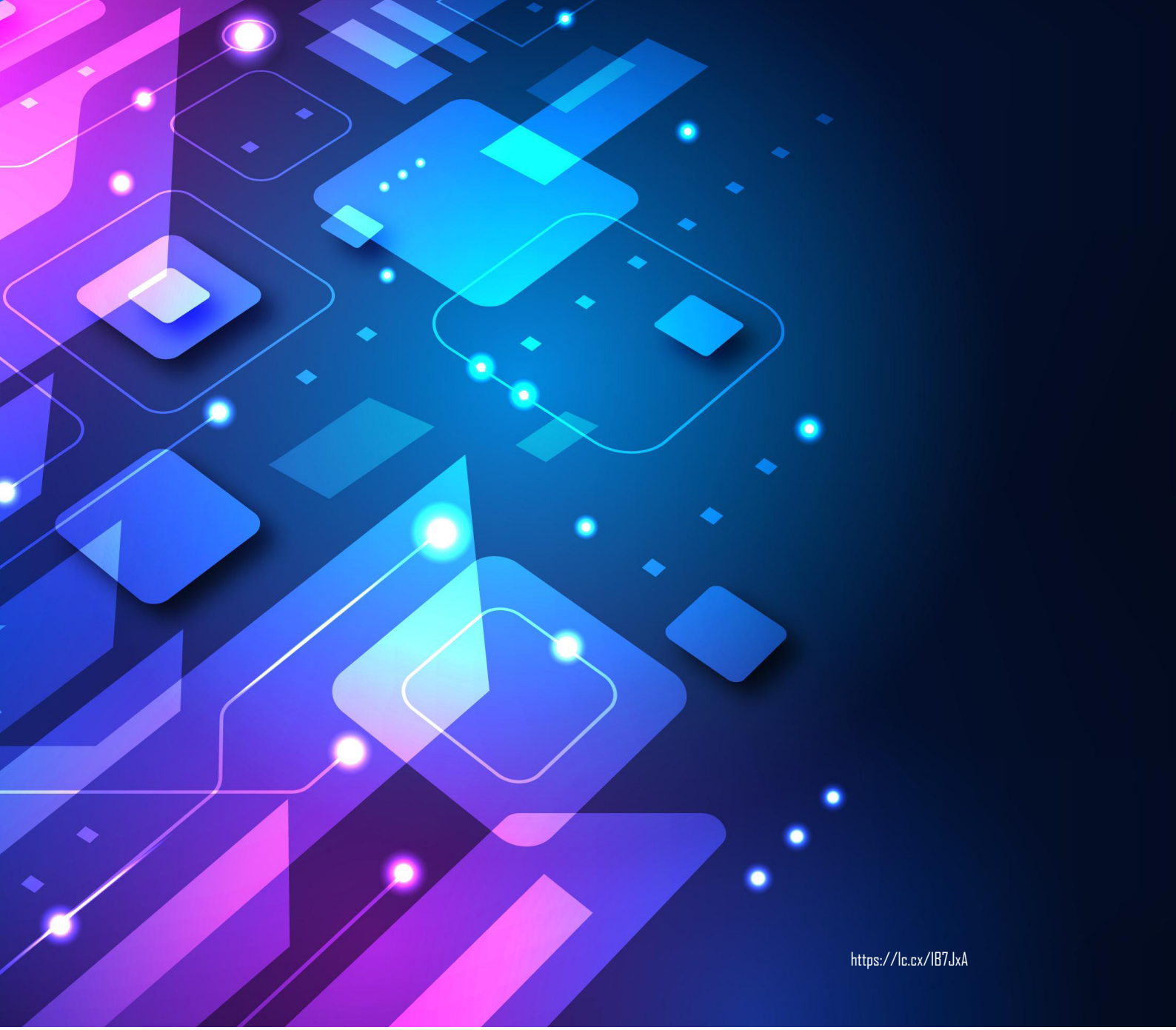
10. **La prueba de la clase de equivalencia tiene varios pasos secuenciales. El orden correcto es:**

    Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos inválidas. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida. Si una condición de entrada es booleana, se define una clase de equivalencia válida y una inválida.

    Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas. Si una condición de entrada específica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida. Si una condición de entrada es booleana, se define una clase de equivalencia válida y una inválida.

    Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos inválidas. Si una condición de entrada

requiere un valor específico, se define una clase de equivalencia válida y dos inválidas. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una inválida y si una condición de entrada es booleana, se define una clase de equivalencia válida y una inválida



CAPÍTULO VIII

Metodologías de desarrollo de software

Definiciones y características

Desde la perspectiva de la Ingeniería de Software, una metodología debe entenderse como el método que permite la organización de un proyecto, esto es, el qué y el cómo del proyecto. La metodología, entonces, devela el orden en el que tienen que realizarse las distintas actividades, al tiempo que provee de las herramientas necesarias para su correcta ejecución (García, 2018).

Con la aplicación de una metodología, se intenta encarar la respuesta a las necesidades típicas de los proyectos como son: el uso de las mejores aplicaciones, emplear el correcto proceso de desarrollo y establecer un proceso estándar en una organización (Piattini, 2016).

Con este antecedente, se puede definir la metodología de la Ingeniería de Software como "un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas" (García, 2018).

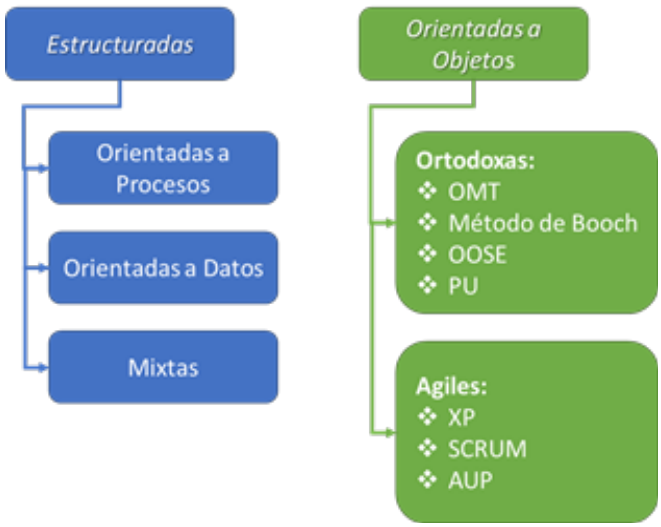
Dentro de este marco, las características deseables en una metodología de Ingeniería de Software son: un proceso de ciclo de vida completo que comprenda aspectos, tanto del negocio como técnicos; un conjunto total de conceptos y modelos que sean internamente consistentes; una colección de reglas y de guías; una descripción total de artefactos a desarrollar; una notación con la cual trabajar que idealmente esté soportada por diferentes herramientas case; un conjunto de técnicas probadas, y un conjunto de métricas junto con el asesoramiento sobre calidad, estándares y estrategias de pruebas (Tesuva, 2014).

Clasificación de las metodologías de desarrollo estructuradas y orientadas a objetos

Las metodologías de desarrollo, tal como se indica en la Figura 31, se clasifican en dos tipos: estructuradas y orientadas a objetos. Las primeras se dividen en: orientadas a procesos, orientadas a datos y mixtas. Las segundas son ortodoxas, aquí encontramos metodologías como OMT, Método de Booch, OOSE PU y en esta misma categoría están las ágiles como XP, SCRUM, AUP. (Hinojosa, Introducción a la ingeniería del software, 2019).

Figura 31

Clasificación de las metodologías de desarrollo



Nota. Clasificación de las metodologías de desarrollo. Adaptado Metodologías de desarrollo. Hinojosa, https://es.scribd.com/document/462507105/Introduccion-Ingenieria-de-Software

Las metodologías orientadas a procesos se fundamentan en el modelo básico entrada-proceso-salida de un sistema, esto es, centradas en la parte del proceso. Su enfoque es la descomposición Top-Down de arriba hacia abajo. Ejemplos de este tipo de metodologías son: Merise, SSADM (Structured Systems Analysis and Design Method), METRICA V3.0 (Ortiz Herrera, 2001).

Las metodologías orientadas a datos se concentran en mayor grado en la entrada-salida. Las actividades de análisis comienzan evaluando los datos y sus interrelaciones para, posteriormente, centrarse en su arquitectura, solo entonces se pueden definir las salidas a producir y los procesos y entradas necesarias para obtenerlas. Por ejemplo, JSP (Jackson Structured Programming), JSD (Jackson Structured Design), LESD (Desarrollo de Sistemas Estructurados) denominado también Warnierr-Orr. (García, 2018).

Las metodologías orientadas a objetos se fundamentan en la integración de dos aspectos de los sistemas de información: datos-procesos. El paradigma usado se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes; el objeto encapsula datos y sus operaciones.

Estas metodologías acortan la distancia existente entre los espacios de conceptos y el de diseño e implementación. Por ejemplo, Metodologías dirigidas por los datos: OMT (Object Modelling Technique) (García, 2018).

Las metodologías ágiles se combinan para una entrega rápida del software. El producto software se desarrolla y se entrega en incrementos, se minimizan así la documentación del proceso y la burocracia. El foco de desarrollo está en el código en sí y no en los documentos de apoyo, tal como se puede obtener de Software Engineering 9/ Figures (Somerville, 2011).

Las metodologías ágiles más conocidas son: XP, SCRUM y AUP. En la Programación Extrema (XP) como se indica en la Figura 32, los requisitos se expresan como escenarios o historias de usuario (HU). Estas HU son implementadas como una serie de tareas que inician con su selección, a continuación, se detallan, se planifica su liberación, se desarrollan e integran y, finalmente, ponen a prueba cada una liberando el producto software y evaluándolo antes de seleccionar otra HU y empezar nuevamente el ciclo de liberación XP. En esta metodología ágil, los programadores trabajan en pares para desarrollar pruebas y luego el código para cada tarea a ser desarrollada.

Figura 32

Ciclo de liberación extrema de XP

Nota. Programación extrema. Adaptado de XP para producir un incremento del sistema por desarrollar https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Recursos complementarios 8

Consulte los siguientes enlaces para reforzar los conocimientos adquiridos:

Accede aquí:

Actividades de aprendizaje 8

En el capítulo 8 (Metodologías de desarrollo de software) puedes observar los diferentes modelos y metodologías explicados de forma breve y eficaz.

Autoevaluación del capítulo 8

1. **La clasificación de las metodologías de desarrollo de software es:**

   Estructuradas/Orientadas a objetos.

   Orientadas a procesos/Orientadas a datos.

   Mixtas.

2. **Las metodologías orientadas a procesos se fundamentan en:**

   El modelo básico de entrada-salida.

   El modelo básico del proceso.

   El modelo básico de entrada-proceso-salida.

3. **El proceso unificado de desarrollo se basa en características que son:**

   Centrado en el diseño/Centrado en la arquitectura/Dirigido por casos de uso.

   Centrado en la arquitectura/Dirigido por casos de uso/Iterativo e incremental.

   Centrado en la arquitectura/Dirigido por casos de uso/Exige poca documentación.

4. **Las metodologías orientadas a datos se concentran en:**

   Entrada-salida.

   Actividades de análisis.

   Procesos de entrada y salidas.

5. **Las metodologías orientadas a objetos se fundamentan en:**

   Datos y procesos.

   Procesos de entrada y salidas.

   Conjunto de objetos.

6. **El diseño y la implementación de las metodologías orientadas a objetos está centrado en:**

   El comportamiento y la estructura.

   Los procesos y los datos.

   Los objetos.

7. **Las metodologías ágiles se clasifican en:**

   Agile Unified Process-AUP.

   Rational Unified Process-RUP.

   Metodología de Edward Yourdon.

8. **Las metodologías orientadas a datos se clasifican en:**

   Jackson Structured Programming-JSP-AUP.

   Jackson Structured Design –JSD-RUP.

   Desarrollo de sistemas estructurados-Warnierr-Orr.

9. **Las metodologías ágiles más conocidas son:**

   OMT, OOSE, PU.

   XP, SCRUM, AUP.

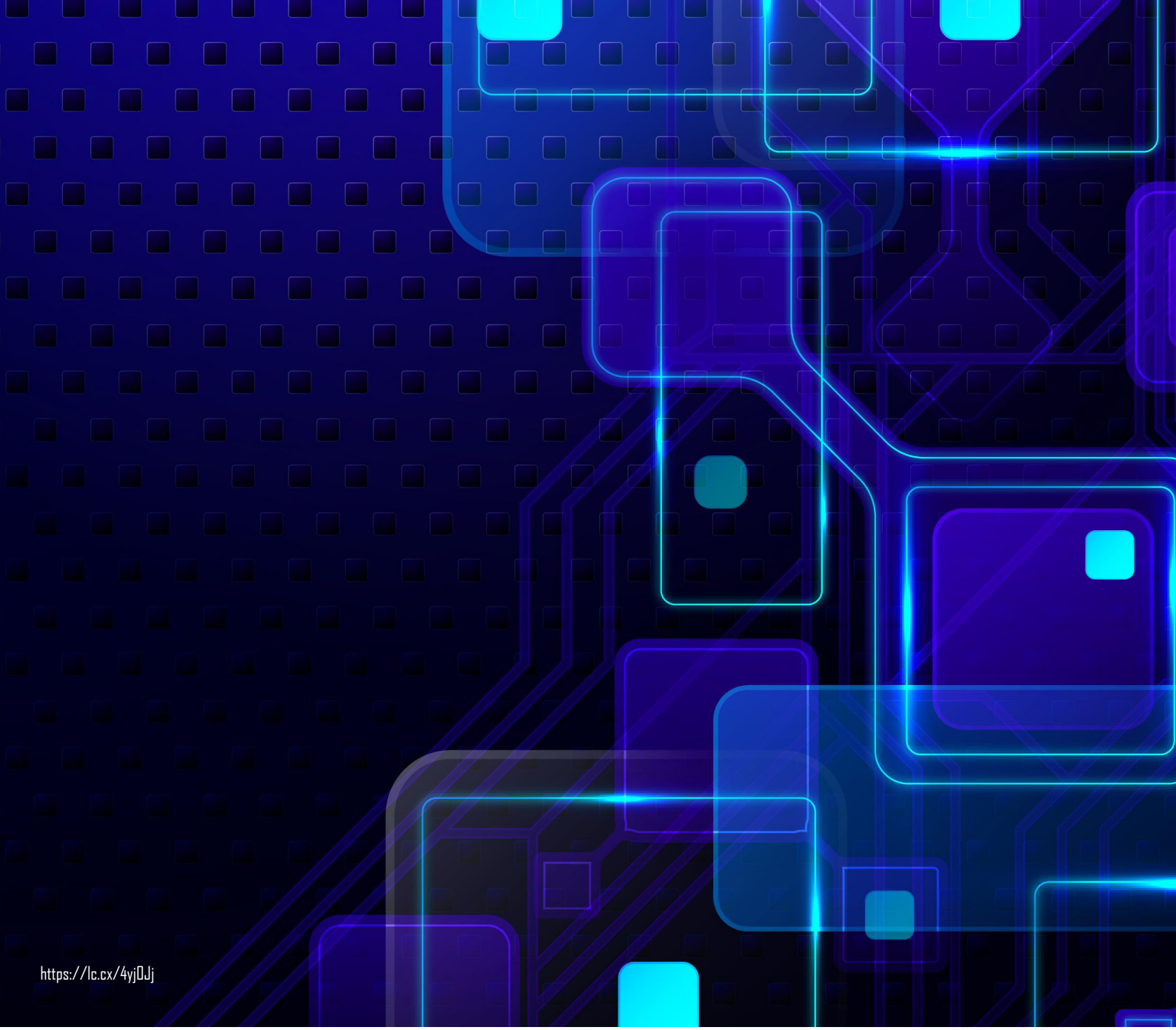
   Método de BOOCH.

10. **Las metodologías orientadas a objetos se clasifican en:**

    Object Modelling Technique-OMT/OOSE/PU/BOOCH.

    Jackson Structured Design –JSD-RUP.

    Metodología de Edward Yourdon



CAPÍTULO IX

Metodologías de desarrollo del software RUP

Introducción al proceso de desarrollo software RUP

El proceso unificado racional (RUP Rational Unified Process) es el mejor ejemplo de cómo un modelo de proceso moderno se derivó del trabajo sobre UML (Unified Modeling Language); el proceso asociado de desarrollo de software unificado RUP está dirigido por los casos de uso y centrado en la arquitectura, es iterativo e incremental (Jacobson & Booch, 2000).

Definición de RUP

Entre las metodologías orientadas a objetos se encuentra el proceso unificado racional (RUP Rational Unified Process), que explora tres elementos: una perspectiva dinámica, que presenta las fases de este modelo a través de una línea de tiempo; una perspectiva estática que presenta las actividades del proceso que se establecen, y finalmente, una perspectiva práctica, que sugiere buenas prácticas a usar durante el proceso de desarrollo de un producto software (Somerville, 2011).

En la figura 33, se presenta un esquema secuencial de las fases que lo componen, estas son:

Concepción, llamada también inicio, cuya meta es establecer un caso empresarial para el producto software, identificando las entidades externas que bien pueden ser otros sistemas o productos software los cuales interactúan con este

Elaboración, su objetivo es desarrollar la comprensión del problema en el entorno del dominio, es decir, entender y abstraer del entorno el problema a resolver; diseñar un plan que permita establecer qué actividades, responsables y recursos serán necesarios y establecer la arquitectura del producto software.

Los entregables generados en esta fase son: un modelo de requisitos para el sistema, conocido como especificación de requisitos de software, así como los modelos UML de estos requisitos, entre los que se cuentan a los diagramas de casos de uso (Jacobson & Booch, 2000).

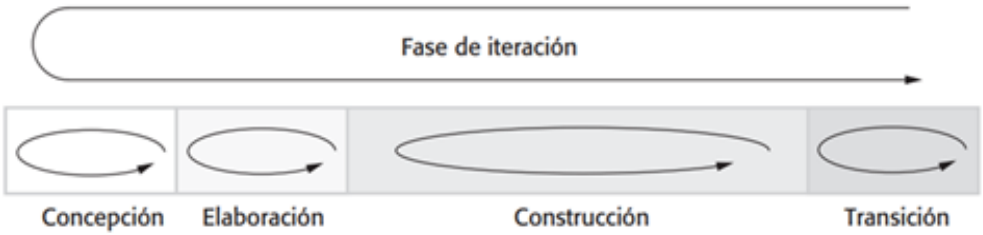
Construcción esta fase desarrolla, diseña, programa y prueba el producto software, al terminarla, se entrega un producto software funcional, así como la documentación para el usuario, por ejemplo, manuales de instalación y de usuario (Jacobson & Booch, 2000).

Transición, define el funcionamiento del producto software en un ambiente real, considerando entre otros aspectos la capacitación y el entrenamiento del personal en los diferentes niveles de uso y las pruebas de funcionamiento pertinentes (Jacobson & Booch, 2000).

La iteración de RUP tiene como apoyo dos formas, cada una de las fases mencionadas anteriormente se presenta en una forma iterativa, tal como se muestra en la Figura 33. Por otro lado, cada iteración realiza una secuencia de disciplinas o flujos de trabajo como son: modelado del negocio, requerimientos, análisis y diseño, codificación, pruebas, instalación, y las disciplinas de soporte entre las que encontramos a la administración de la configuración y cambios, y la administración de proyectos y ambiente (Jacobson & Booch, 2000).

Figura 33

Fases en el proceso unificado racional (RUP)



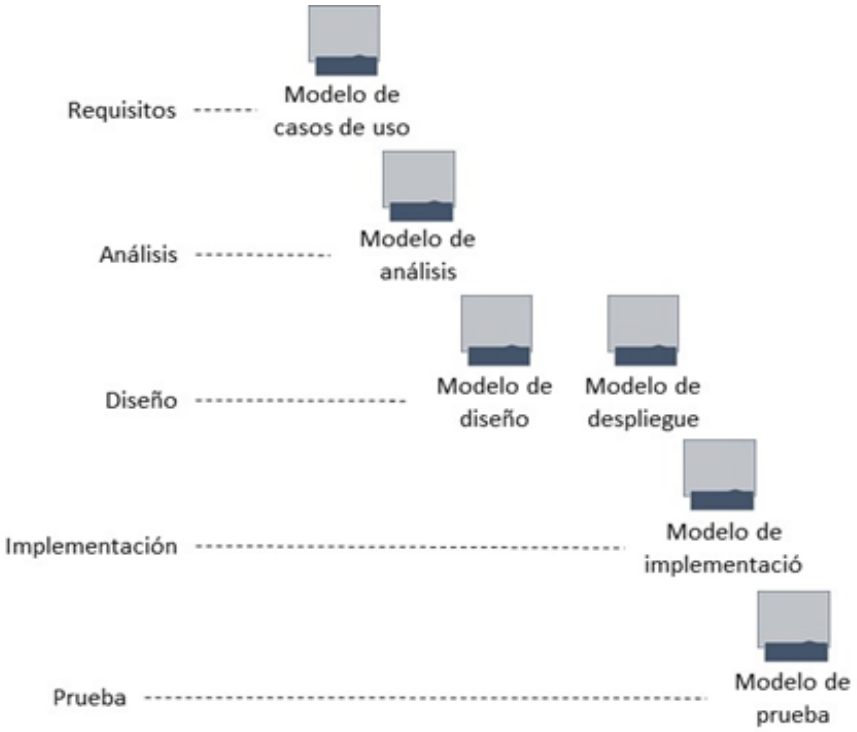
Nota. Adaptado del Proceso Unificado Racional https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

RUP dirigido por casos de uso

En la Figura 34, el proceso de desarrollo comienza extrayendo los requisitos de los usuarios y los convierte en modelos de casos de uso, luego analizan y diseñan el producto software para convertirlo en modelos de casos de uso (Jacobson & Booch, 2000).

Figura 34

Flujos de trabajo del proceso unificado racional RUP



Nota. El proceso Unificado de Desarrollo de Software. Adaptado de El proceso Unificado de Desarrollo de Software. https://www.academia.edu/11946867/El_Proceso_Unificado_de_desarrollo_de_sofware?email_work_card=-view-paper

La captura de requisitos tiene dos objetivos. El primero de ellos es identificar los requisitos funcionales y, el segundo, representarlos de la mejor manera para los usuarios y los desarrolladores. Frecuentemente, las razones que apoyan el utilizar casos de uso (CU) se debe a que estos proporcionan un medio sistemático e intuitivo para liberar requisitos funcionales; por otro lado, estos CU son el punto de partida para el proceso de desarrollo de un producto software (Jacobson & Booch, 2000).

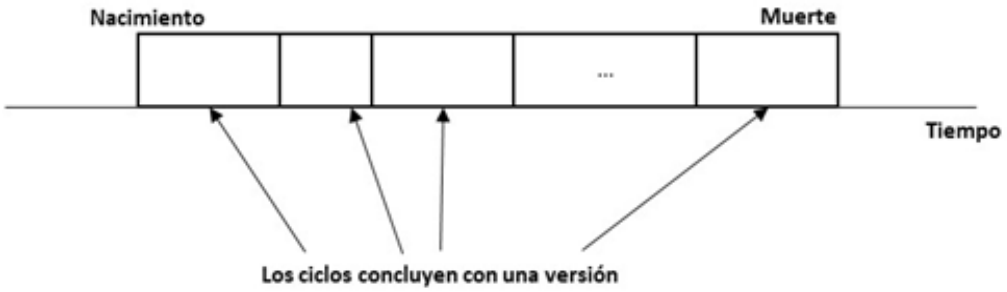
El ciclo de vida de RUP

RUP se repite a lo largo de una serie de iteraciones o ciclos cortos que constituyen la vida de un producto software. En la Figura 35, cada ciclo entrega un prototipo funcional del producto para los usuarios. Cada iteración consta

de cuatro fases: concepción/inicio, elaboración, construcción y transición, tal como se muestra en la Figura 36 (Jacobson & Booch, 2000).

Figura 35

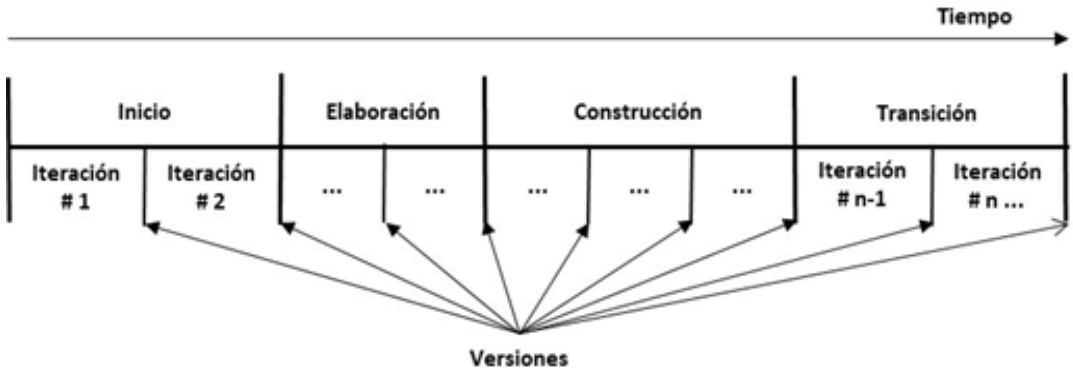
Ciclo de vida de RUP



Nota. El proceso Unificado de Desarrollo de Software. Adaptado de El proceso Unificado de Desarrollo de Software. https://www.academia.edu/11946867/El_Proceso_Unificado_de_desarrollo_de_sofware?email_work_card=-view-paper

Figura 36

Un ciclo corto con sus fases e iteraciones



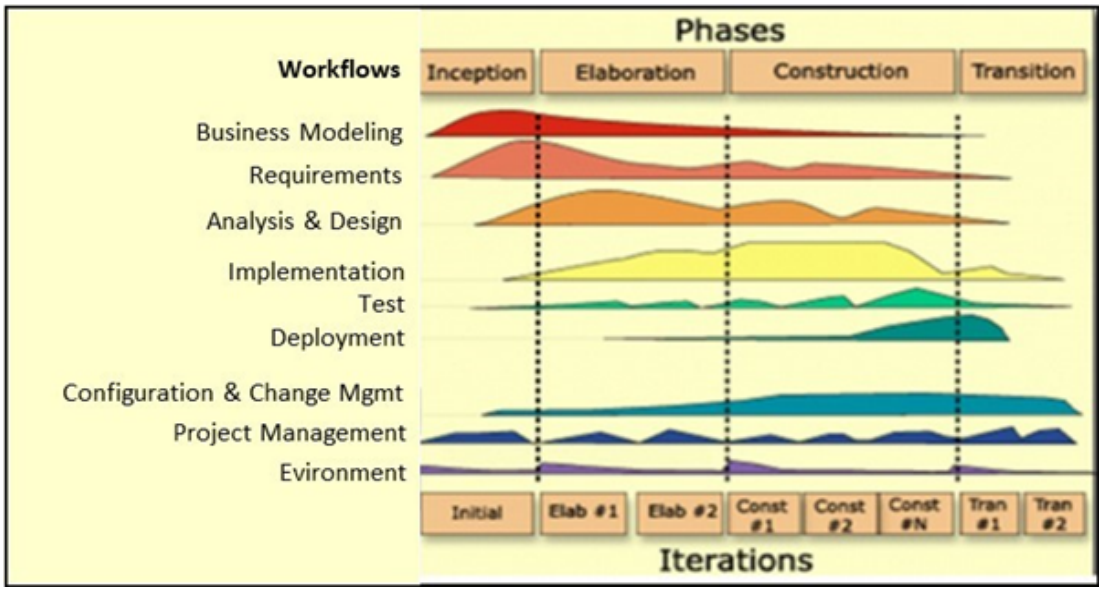
Nota. El proceso Unificado de Desarrollo de Software. Adaptado de El proceso Unificado de Desarrollo de Software. https://www.academia.edu/11946867/El_Proceso_Unificado_de_desarrollo_de_sofware?email_work_card=-view-paper

En la Figura 37, cada iteración o ciclo se desarrolla a lo largo del tiempo y se divide en cuatro fases. Los primeros se muestran en la columna de la izquierda, es decir, flujos de trabajo como requisitos, análisis, diseño, implementación y prueba; las curvas que se muestran a la derecha son solo aproximaciones gráficas que denotan hasta dónde pueden llegar los flujos de trabajo. Una ite-

ración pasa por los cinco flujos de trabajo mencionados en párrafos anteriores (Jacobson & Booch, 2000).

Figura 37

Flujos de trabajo con sus disciplinas y las fases de RUP



Nota. El proceso Unificado de Desarrollo de Software. Adaptado de El proceso Unificado de Desarrollo de Software. https://www.academia.edu/11946867/El_Proceso_Unificado_de_desarrollo_de_sofware?email_work_card=-view-paper

Recursos complementarios 9

Consulta el material para reforzar los conocimientos adquiridos

Accede aquí:

Actividad de aprendizaje 9

1. **Junte los objetivos con la correspondiente fase del Proceso Unificado de Desarrollo.**

1. Lograr versiones útiles (alfa, beta y otras versiones de prueba) tan rápido como sea práctico	A. Inicio/Elaboración/Construcción/ Transición
2. Lograr que las partes interesadas estén de acuerdo en que las líneas de base de implementación están completas	B. Inicio/Elaboración/Construcción/ Transición
3. Discriminar los casos de uso críticos del sistema, que son los escenarios principales de operación que impulsarán las principales compensaciones de diseño	C. Inicio/Elaboración/Construcción/ Transición
4. Lograr la autosuficiencia del usuario.	D. Inicio/Elaboración/Construcción/ Transición

2. **Seleccione la opción correcta. El Proceso Unificado de Desarrollo es. Seleccione una:**

   Metodología para el desarrollo de software que define claramente: quién, cómo, cuándo y qué debe hacerse en el proyecto

   Programa para desarrollar software con poca documentación, que permite el cambio ágil dentro del proyecto

   Herramienta que permite el desarrollo de software avanzado, sin necesidad de datos específicos.

   Metodología ágil para el desarrollo de software

3. **Seleccione las opciones correctas. El Proceso Unificado de Desarrollo se basa en las siguientes características fundamentales: Seleccione una o más de una:**

   Exige poca documentación

   Dirigido por casos de uso

   Proceso secuencial

   Iterativo e incremental

   Centrado en la arquitectura

   Centrado en el diseño

4. **Una metodología de desarrollo de software es un conjunto de técnicas y _____en fases para el desarrollo de _____, de manera eficaz, y**

abarca el _____del mismo. Es una colección _____ para la resolución de una clase de problemas. Las metodologías de desarrollo de software descomponen el proceso en actividades

Procedimientos organizados

Productos software

Ciclo de vida

Métodos

5. **En las siguientes oraciones complete con el término correspondiente:**

_____es un conjunto de técnicas y procedimientos organizados en fases para el desarrollo de productos software, de manera eficaz, y abarca el ciclo de vida del mismo.

_____es una técnica repetible para la resolución de un problema específico.

_____es un conjunto de reglas gráficas o textuales para representar un modelo

Proceso

Metodología

Notación

Método

6. **Seleccione los términos correctos. RUP tiene dos dimensiones:**

1. El eje_____ representa _____y muestra los aspectos del _____ del proceso a medida que se desarrolla en iteraciones.	A. Horizontal/el tiempo/ ciclo de vida/
2. 1. El eje_____ representa _____ como requisitos, análisis y diseño, implementación, que lógicamente agrupan _____ por afinidad	B. Vertical/las disciplinas/las actividades/

7. Empareje las disciplinas con su definición.

1. Pone el sistema a disposición de los usuarios finales	A. Entrega/Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración.
2. Administrar el acceso a los artefactos del proyecto y controla y gestiona los cambios	B. Entrega/Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración
3. Comprende el negocio y el dominio del problema y presenta una solución viable	C. Entrega/Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración.
4. Gestionar riesgos y dirige y coordinar personas	D. Entrega/Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración
5. Asegura la calidad verifica que los requisitos se cumplan	E. Entrega/Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración
6. Asegurar que el equipo cuente con lo necesario, orientación y herramientas adecuados	F. Entrega/Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración
7. Transforma los modelos en código fuente	G. Implementación/Modelamiento/Gestión de Proyecto/Prueba/Entorno/Entrega/Gestión de la configuración.

8. Mira el siguiente vídeo

Autoevaluación del capítulo 9

1. **Las fases que conforman RUP son:**

   Planificación, diseño, codificación, evaluación.

   Concepción, elaboración construcción, transición.

   Análisis, diseño, desarrollo, pruebas.

2. **RUP se maneja en el proceso de desarrollo:**

   Como un proceso lineal.

   Con iteraciones.

   Con el uso de Sprints.

3. **La fase de elaboración consiste en:**

   Desarrollar la comprensión del problema.

   Definir el funcionamiento del producto software.

   Desarrollar, diseñar y programar.

4. **La fase de construcción consiste en:**

   Desarrollar la comprensión del problema.

   Definir el funcionamiento del producto software.

   Desarrollar, diseñar y programar.

5. **La fase de transición consiste en:**

   Desarrollar la comprensión del problema.

   Definir el funcionamiento del producto software en un ambiente real.

   Desarrollar, diseñar y programar.

6. **RUP dirigido por casos de uso tiene flujos de procesos que son:**

   Requisitos, análisis, diseño, implementación y pruebas.

   Modelo de casos de uso, modelo de análisis, modelo de despliegue.

   Modelo de implementación, modelos de diseño, modelo de pruebas.

7. **RUP tiene un ciclo de vida que repite las fases ordenadas como:**

   Concepción, elaboración, construcción y transición.

   Construcción concepción, elaboración, y transición.

   Elaboración, concepción, construcción y transición.

8. **Un ciclo corto con sus fases e iteraciones tiene:**

   Iteraciones no finitas.

   Iteraciones finitas y no finitas.

   Iteraciones finitas que generan diferentes versiones.

9. **Los flujos de trabajo de RUP son:**

   Inicio, elaboración, construcción, transición.

   Requisitos, elaboración, construcción, transición.

   Inicio, elaboración, construcción, prueba.

10. **Un objetivo de la captura de requisitos en RUP e**s:

    Identificar los requisitos no funcionales y su representación.

    Identificar los requisitos funcionales y no funcionales y su representación.

    Identificar los requisitos funcionales y su representación

CAPÍTULO X

Metodologías ágiles para el desarrollo de software

Métodos ágiles: XP, AUP

Los procesos ágiles de desarrollo de software se diseñan para producir de manera rápida un producto software funcional, este no se desarrolla como un todo sino como un conjunto de varios incrementos, donde cada uno de los cuales termina construyendo una nueva funcionalidad del producto (Somerville, 2011).

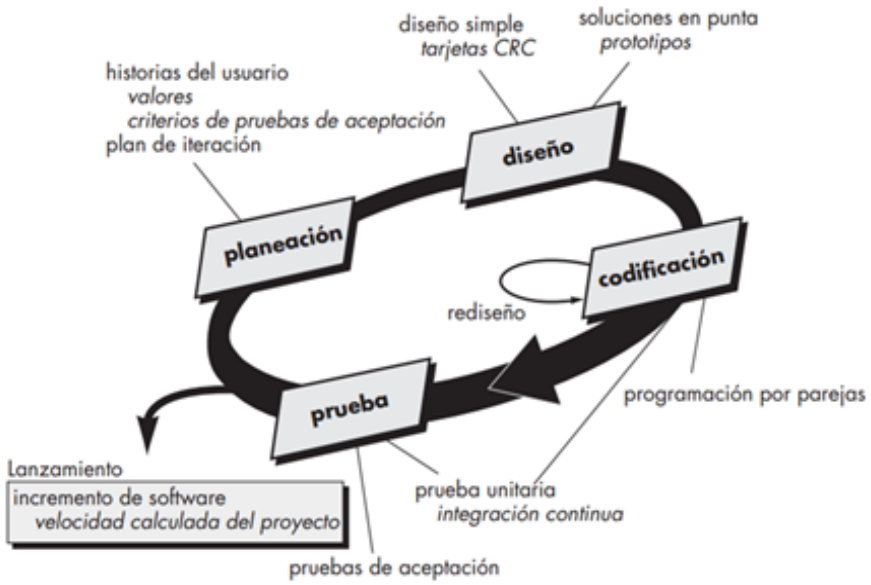
Las características fundamentales que requieren los métodos ágiles como Extreme Programming (XP) y Agile Unified Process (AUP) en ocasiones son difíciles de cumplir por varias razones, entre ellas:

1. No siempre se puede contar con la participación de un cliente que disponga y desee pasar con el equipo de desarrollo todo el tiempo que este lo necesite.
2. Disponer de programadores con la personalidad necesaria que permitan una participación intensa con el cliente.
3. Priorizar los cambios que se generan a nivel de requisitos funcionales y su respectiva documentación.
4. Mantener el trabajo a desarrollar en el proceso de la manera más simple posible.
5. Conservar una cultura de calidad frente al control de cambios y en la adaptación a estos (Somerville, 2011).

La programación extrema (XP) como término fue acuñada por Beck (2000) y fue circunscrita a un proceso de desarrollo iterativo; de esta manera, el enfoque es orientado a objetos, dado que reúne un conjunto de reglas y prácticas que ocurren en el marco de cuatro actividades estructurales: planeación, diseño, codificación y pruebas, Figura 38 (Pressman, 2005).

Figura 38

El proceso de programación extrema: fases e iteración



Nota. Definición y análisis de la programación extrema (XP) Adaptado del proceso XP https://docs.google.com/document/d/1x1uFkX13aWHfVkssP-TqSZgRDN-b0GiYVXuSScnQ3YCs/edit?usp=sharing

El proceso unificado ágil (AUP) conserva una filosofía "en serie para lo grande" e "iterativa para lo pequeño". Está basado en RUP pues acoge las fases clásicas de concepción, elaboración, construcción y transición; de manera que provee una secuencia de actividades de Ingeniería de Software que resulta muy significativa para los clientes, ya que observan en un solo flujo el desarrollo de un producto software; por otro lado, cada actividad de este proceso incluye iteraciones para la entrega ágil deseada (Pressman, 2005).

SCRUM es un método ágil general, se basa en la administración iterativa del desarrollo de un producto software, pero no en los enfoques técnicos de la Ingeniería de Software. La Figura 39 representa un diagrama del proceso de administración de Scrum, donde se pueden apreciar las siguientes fases.

1. Planeación del bosquejo, enmarcado por la definición de la arquitectura de software y los objetivos del proyecto.
2. Ciclos cortos o Sprint, donde cada ciclo desarrolla un incremento.
3. Cierre, donde termina el proyecto y entrega la documentación, por ejemplo, manual de instalación y manual de usuario del producto software (Pressman, 2005).

Figura 39

El proceso de Scrum y el ciclo sprint

Nota. Administración de un proyecto ágil. Adaptado de diagrama de proceso de administración de SCRUM. https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Principios de los métodos ágiles

En la tabla 2, los tres métodos ágiles (XP, AUP y Scrum), mencionados anteriormente, comparten una serie de principios de acuerdo con el manifiesto ágil, por ende, comparten muchas de estas características; sin embargo, sus procesos de desarrollo son diferentes al lograr un mismo objetivo, esto es, desarrollar un producto software (Somerville, 2011).

Tabla 2

Los principios de los métodos ágiles y su descripción

Principio	Descripción
Partición del cliente	Los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.
Entrega incremental	El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.
Personas no procesos	Tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar son procesos establecidos.

Adoptar el cambio	Esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios
Mantener simplicidad	Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.

Nota. Métodos ágiles. Adaptado de los principios, según el manifiesto ágil. https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Desarrollo rápido dirigido por un plan y desarrollo ágil SCRUM

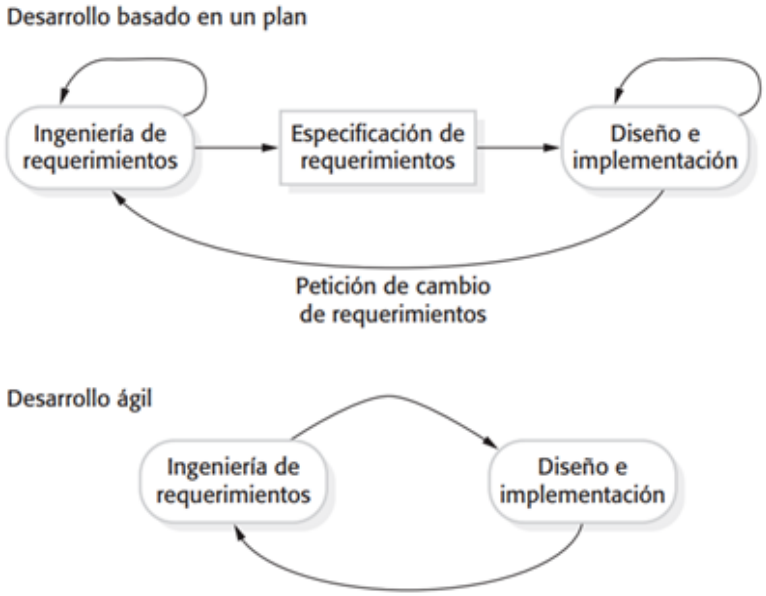
Un proceso de desarrollo ágil dirigido por un plan atiende el desarrollo y la entrega por incrementos, en él se pueden asignar requisitos funcionales, así como planificar el diseño y el desarrollo como una serie de ciclos cortos llamados Sprint, donde un Sprint es un tiempo no más allá de una a dos semanas en el que se debe desarrollar un producto software (Somerville, 2011).

La experiencia en el desarrollo de proyectos establece que, para generar un producto software, el punto de partida es la identificación clara del problema que se desea resolver y la planificación a través de un perfil de proyecto donde se describa el objetivo general, al menos tres secundarios, el alcance y el cronograma para empezar.

El desarrollo basado en un plan de la Figura 40 inicia con la Ingeniería de requerimientos, continúa con la especificación, para luego ir al diseño e implementación (Somerville, 2011).

Figura 40

Especificación ágil y dirigida por un plan de desarrollo ágil



Nota. Métodos ágiles. Adaptado de desarrollo dirigido por un plan y desarrollo ágil https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

En la práctica, la dificultad radica en que usted como estudiante y futuro ingeniero, debe tener claro qué es un requisito funcional y cómo identificarlo al momento de cumplir con el ciclo de desarrollo ágil, es decir, partir de la Ingeniería de requisitos, continuar con la especificación, pasar al diseño y, finalmente, a la implementación. Recuerde que el verbo implementar según la Real Academia Española, (RAE) textualmente significa "Poner en funcionamiento o aplicar métodos, medidas, etc., para llevar algo a cabo" (Real Academia Española, 2020).

Recursos complementarios 10

Consulta el material para reforzar los conocimientos adquiridos

Accede aquí:

Actividad de aprendizaje 10

En el capítulo 10 se contrasta los tipos de metodologías ágiles de desarrollo de software.

Puedes revisar el siguiente video como un resumen.

Autoevaluación del capítulo 10

1. **Una característica de los métodos ágiles es:**

   No siempre pueden contar con la participación del cliente.

   Son metodologías secuenciales que no consideran iteraciones.

   La volatilidad de los requisitos no es considerada en estas metodologías ágiles.

2. **La programación extrema (XP) como término fue acuñada por Beck en 2000.**

   Cierto.

   Falso.

   Ninguna de las opciones.

3. **En el proceso de programación extrema, el lanzamiento es:**

   Cuando todas las fases (planeación, diseño, codificación y pruebas) quedan suspendidas.

   Cuando ninguna de las fases (planeación, diseño, codificación y pruebas) terminan con un incremento del producto software.

   Cuando solo las fases planeación y diseño terminan con un incremento del producto software.

4. **El proceso unificado ágil (AUP) no está basado en RUP.**

   Cierto.

   Falso.

Ninguna de las opciones.

5. **SCRUM es un método ágil general basado en:**

   La administración iterativa del desarrollo de un producto software.

   Los enfoques técnicos de la ingeniería de software.

   Las dos opciones.

6. **El principio de participación del cliente consiste en:**

   Los clientes deben intervenir conjuntamente con el equipo de desarrollo durante el proceso.

   Los clientes no deben intervenir conjuntamente con el equipo de desarrollo durante el proceso.

   No priorizar nuevos requerimientos sino mantener los existentes sin cambio alguno.

7. **El principio de mantener simplicidad consiste en:**

   Enfocarse en la complejidad tanto en el software a desarrollar como en el proceso de desarrollo.

   Esperar que cambien los requerimientos del sistema y adoptar estos cambios.

   Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo.

8. **Un sprint es un tiempo no más allá de una a dos semanas en las que se debe desarrollar un producto software.**

   Cierto.

   Falso

   Ninguna de las opciones.

9. **El punto de partida en la experiencia de desarrollo de un producto software es:**

   Identificación clara del problema.

   Planificar y realizar un perfil de proyecto.

   Las dos opciones.

10. En la práctica, cuáles son los conocimientos que usted necesita como futuro ingeniero en el desarrollo de un producto software.

Tener claro qué es un requisito funcional.

Conocer de especificación de estos requisitos, diseñar e implementar.

Las dos opciones.

CAPÍTULO XI

Desarrollo guiado por pruebas de caja blanca y caja negra

Técnicas de caja blanca

Las pruebas consumen tiempo y son costosas. Su eficacia en cuanto a la elección de casos de prueba de unidad implica que estos casos de prueba deben demostrar que el componente utilizado responde de buena forma frente al requisito para el cual fue preparado. En el caso de existir defectos, estos deben revelarse en el momento de ejecutar el caso de prueba (Somerville, 2011).

El diseño de casos de prueba tiene un objetivo importante: derivar un conjunto de pruebas con mayor probabilidad de descubrir errores en el producto software. Para conseguirlos se usan dos técnicas diferentes y complementarias: pruebas de caja blanca y de caja negra (Pressman, 2005).

La prueba de caja blanca, llamada también Caja de vidrio, usa las estructuras de control, por ejemplo, IF THEN ELSE, utilizadas en programación estructurada como parte del diseño a nivel de componentes, con el objetivo de realizar casos de prueba, de tal forma que al utilizar este tipo de técnicas, se desarrollen casos de pruebas bajo los criterios de:

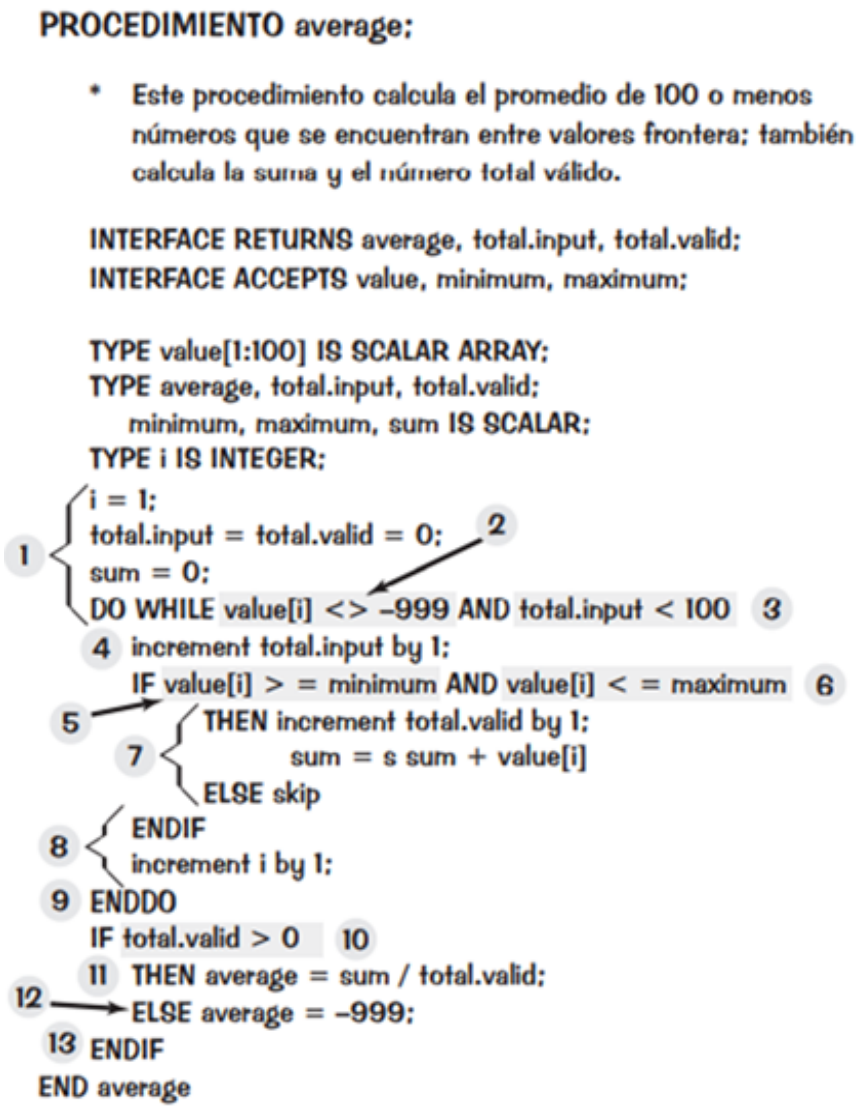
1. Garantizar que todas las rutas sean independientes dentro de un módulo y que estas se revisen por lo menos una vez.
2. Revisar todas las decisiones lógicas (IF THEN ELSE) en sus dos estados (verdadero y falso).
3. Ejecutar todos lo bucles en sus fronteras.
4. Revisar las estructuras de datos internas garantizando su validez. (Pressman, 2005)

Una técnica de prueba de caja blanca es la prueba de ruta o trayectoria básica, propuesta por Tom McCabe. Esta permite al diseñador de casos de prueba realizar una medida de complejidad lógica de un diseño a nivel de código fuente (Pressman, 2005).

En la Figura 41 se inicia el proceso para aplicar la prueba de caja blanca, en este caso, la prueba de ruta o trayectoria básica; en el paso 1, escoja una muestra de código fuente que incluya estructuras de control, por ejemplo, PDL procedimiento Average. Debe identificar los nodos y numerar los bloques de estas estructuras como se indica en este ejemplo (Pressman, 2005).

Figura 41

PDL con identificación de nodos



Nota. Técnicas de pruebas de software. Adaptado de casos de pruebas con PDL, https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

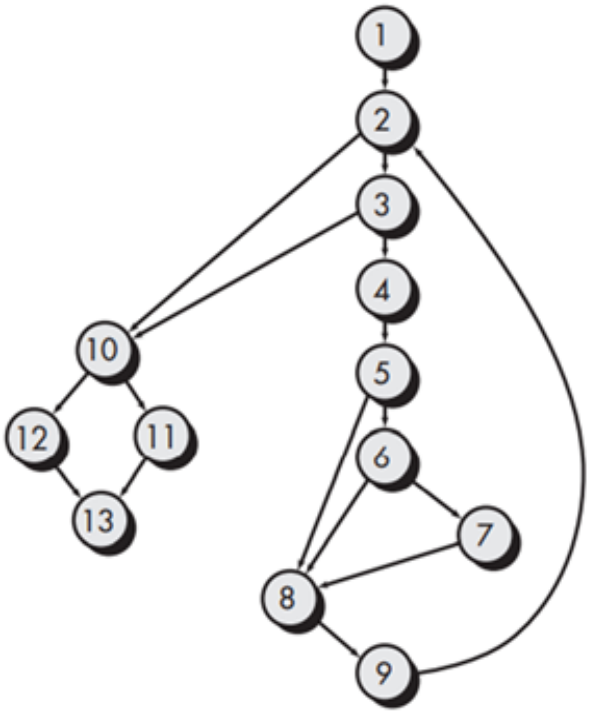
En la Figura 42, paso 2, determine la complejidad ciclomática, V(G) del gráfico de flujo resultante. Existen tres formas de cálculo de V(G):

1. El número de regiones del gráfico del flujo corresponde a V(G).
2. La complejidad ciclomática V(G) para un grafo de flujo G es igual a V(G)=E-N+2 donde E=número de aristas del gráfico del flujo y, N=número de nodos del gráfico del flujo.

3. La complejidad ciclomática V(G) para un gráfico de flujo G también se define como V(G)=P+1 donde P= número de nodos predicados (IF-THEN ELSE) en el gráfico del flujo G.

Figura 42

Gráfico del flujo para el procedimiento average con nodos, aristas y regiones



Nota. Técnicas de pruebas de software. Adaptado de grafos de flujo, https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

El proceso de cálculo de V(G) es el siguiente, de acuerdo con lo mencionado en párrafos anteriores, mediante las fórmulas de complejidad ciclomática:

a) V(G)=6 regiones.

b) V(G)=17 aristas-13 nodos+2=6

c) V(G)= 5 nodos predicados+1=6. (Pressman, 2005)

En la tabla 3, paso 3, determine un conjunto básico de rutas linealmente independientes, donde el valor V(G) proporciona el número de rutas linealmente independientes de las estructuras de control analizadas en el paso 2. Para el caso del ejemplo procedimiento Average se obtienen 6 rutas, por otro lado (…), en las rutas 4,5,6 significa que es aceptable cualquier ruta a través del resto de la estructura de control (Pressman, 2005).

Tabla 3

Rutas números 1,2,3,4,5,6

Ruta número	Descripción
Ruta 1	1-2-10-11-13
Ruta 2	1-2-10-12-13
Ruta 3	1-2-3-10-11-13
Ruta 4	1-2-3-4-5-8-9-2
Ruta 5	1-2-3-4-5-6-8-9-2
Ruta 6	1-2-3-4-5-6-7-8-9-2

Nota. Técnicas de pruebas de software. Adaptado de rutas o caminos para el cálculo de V(G) https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

En el paso 4 prepare casos de prueba que fuercen la ejecución de cada ruta en el conjunto básico de estos datos, estos últimos deben elegirse de forma que las condiciones en los nodos predicado se establezcan de acuerdo a cómo se prueba cada ruta y, una vez que se completen todos los casos de prueba, el Tester puede asegurarse que todas las rutas y sus enunciados se ejecutaron al menos una vez (Pressman, 2005).

La Figura 43 presenta un ejemplo de prueba de caja blanca de un proyecto académico de la Universidad de las Fuerzas Armadas ESPE de un "Sistema de Gestión de Productos y Control de Inventario de un micro mercado", donde aplica estas pruebas en el control de calidad del producto de desarrollo de software.

Figura 43

Sistema de gestión de productos y control de inventario de un micro mercado, caja blanca

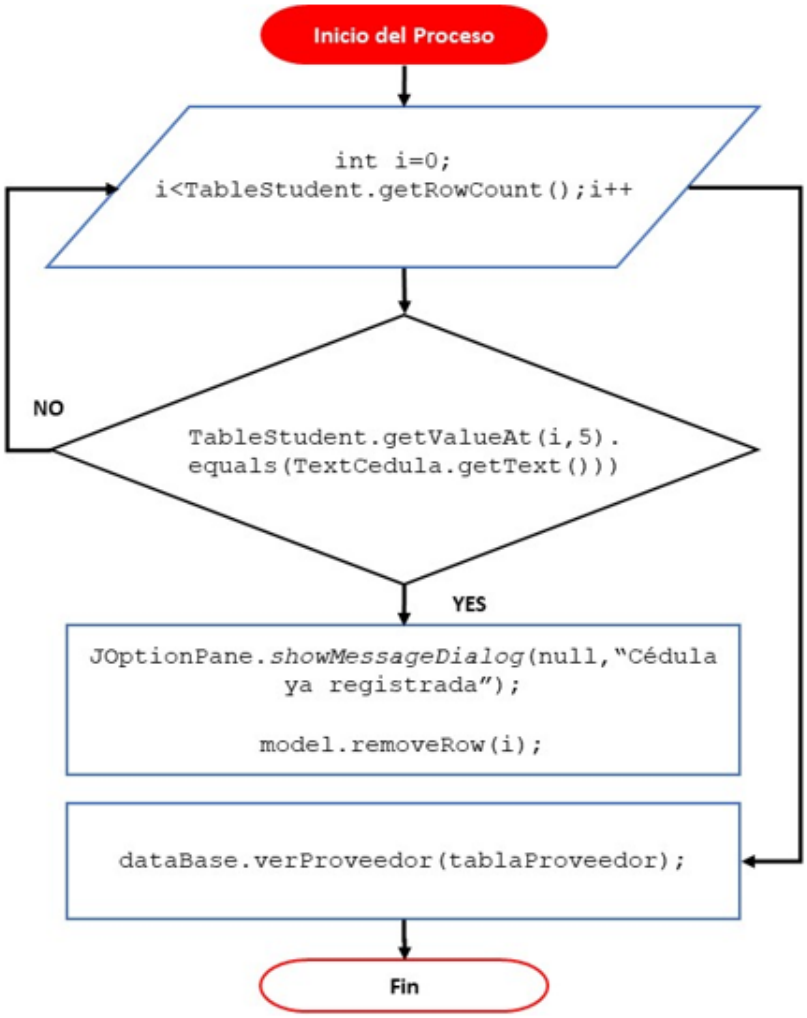
Prueba de Caja Blanca Duplicación de Cédula

Código

```
public void ValidarIngresosTabla(){
      for(int i=0; i<tablaProveedor.getRowCount();i++)
      {
      if(tablaProveedor.getValueAt(i, 5).equals(TextCedula.getText()))
```

```
(
JOptionPane.showMessageDialog(null,"Cédula ya registrada");
model.removeRow(i);
)
)
dataBase.verProveedor(tablaProveedor);
)
```

Diagrama de Flujo

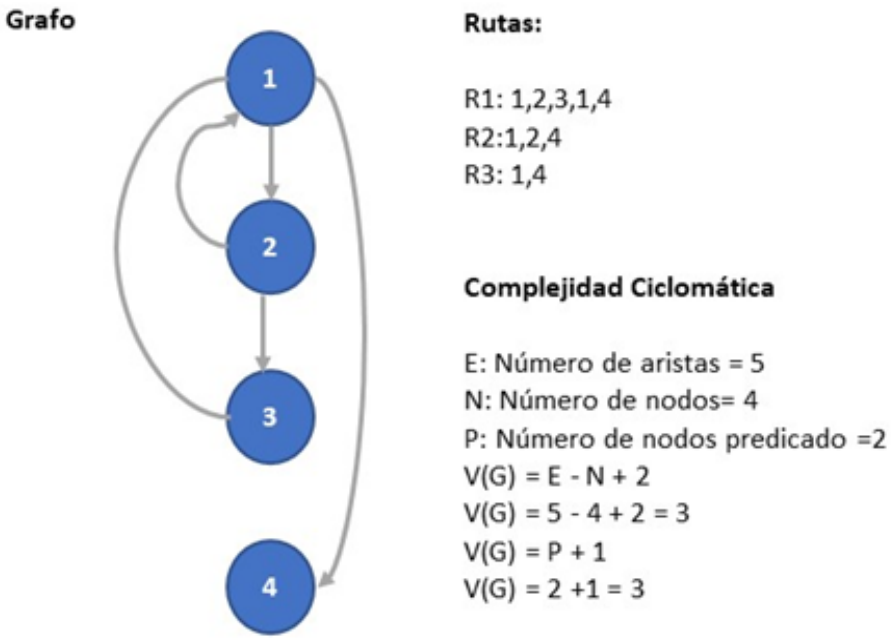


Nota. Prueba de caja blanca. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

La Figura 44 presenta el grafo de flujo, las rutas y el cálculo de la complejidad ciclomática. Adaptado de Achig; Egas; Pillajo; Tavaris, 2021 trabajo curso cuyo desarrollo se referencia en Pruebas de caja blanca y negra (Zap, 2024).

Figura 44

Caja blanca con su grafo de flujo y el cálculo de complejidad ciclomática V(G).



Nota. Prueba de caja blanca, cálculo de complejidad ciclomática V(G). Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Técnicas de caja negra

Las pruebas de caja negra, llamadas también pruebas de comportamiento, se basan en los requisitos funcionales del producto software, es decir, permiten producir casos de prueba bajo un conjunto de condiciones de entrada que verifican completamente el cumplimiento de los requisitos mencionados anteriormente (Pressman, 2005).

El objetivo de estas pruebas es intentar encontrar errores en:

1. Funciones incorrectas o faltantes
2. Errores de interface
3. Errores en las estructuras de datos o en el acceso a bases de datos externas
4. Errores de comportamiento o rendimiento

5. Errores de inicialización y terminación (Pressman, 2005).

La prueba de caja negra conocida como partición de equivalencia divide el dominio de entrada de un programa en clases de datos de los que pueden resultar casos de prueba.

El diseño de casos de prueba para la partición de equivalencia se basa en la evaluación de clases de equivalencia para un conjunto de datos de entrada, estas se pueden definir de acuerdo con los siguientes lineamientos:

1. Si una condición de entrada especifica un rango, defina una clase de equivalencia válida y dos no válidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.
3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.
4. Si una condición de entrada es booleana, es decir valor True (verdadero) o False (falso), se define una clase de equivalencia válida y una no válida.

La tabla 4 presenta un ejemplo de prueba caja blanca de un proyecto académico de la Universidad de las Fuerzas Armadas ESPE, de la asignatura de Metodologías de Desarrollo de Software, mencionada en párrafos anteriores donde se aplica esta prueba en el control de calidad del producto de desarrollo de software.

Tabla 4

Caja negra "Sistema de gestión de productos y control de inventario de un micro mercado".

Variable	Clase de equivalencia	Estado	Representante
Nombres -	EC1: Nombre-Apellido==	Valido	Luis Tavaris
Apellidos	TextNombre, TextApellido		----------
	EC2: Nombre-Apellido ¡=	No	
	TextNombre, TextApellido	válido	
Cédula	EC3: TextCedula=10	Válido	1719314179

Verificación cadena TextBox	EC4: TextCedula>10 EC5: TextCedula<10	No válido No válido	02312312 171931114179
Número celular	EC6: TextCelular=10 EC7: TextCelular>10 EC8: TextCelular<10	Válido No válido No válido	0961549661 0961549661234 0961549
Fecha de nacimiento	EC9:DateOfBrith==forma.dateOfBirth EC10: DateOfBrith ¡=forma.dateOfBirth	Válido No válido	2/08/2021 ----------
Correo electrónico	EC11: email==TextEmail EC12: email¡=TextEmail	Válido No válido	ltavaris@esé.edu.ec ----------
Dirección domiciliaria	EC13: dirección== TextDireccion EC14: dirección¡= TextDireccion	Válido No válido	Av. General Rumiñahui S/N ----------

Nota. Prueba de caja negra, partición de clases de equivalencia. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaO-FO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

En la Figura 45 y Figura 46 se muestran las capturas de pantalla para verificar los requisitos funcionales a nivel de interface para el ingreso de datos en el formulario de registro de usuarios.

Figura 45

Prueba caja negra formulario de registro de usuarios campos completos

Nota. Prueba de caja negra, Formulario de registros de usuarios campos completos. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzD-FxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Figura 46

Prueba caja negra formulario de registro de usuarios campos no completados

Nota. Prueba de caja negra, Formulario de registros de usuarios campos no completados. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Recursos complementarios 11

Consulta el material para reforzar los conocimientos adquiridos

Accede aquí:

Actividad de aprendizaje 11

El tema 11 resume los tipos de pruebas de caja blanca y caja negra.

Puedes revisar el siguiente video para profundizar sobre estas dos técnicas CB y CN:

Accede aquí:

Autoevaluación del capítulo 11

1. **La eficacia de las pruebas consiste en que:**

   El componente utilizado responde de buena forma frente al requisito para el cual fue preparado.

   El componente utilizado responde de mala forma frente al requisito para el cual fue preparado.

   En el caso de existir defectos estos no se muestran en el momento de la ejecución.

2. **Al diseñar casos de prueba el objetivo es derivar un conjunto de pruebas con menor probabilidad de descubrir errores.**

   Cierto.

Falso

Ninguna de las opciones.

3. **La prueba de caja blanca se conoce también como caja de vidrio.**

   Cierto.

   Falso.

   Ninguna de las opciones.

4. **Una de las técnicas de las pruebas de caja blanca es:**

   Prueba de ruta o trayectoria.

   Complejidad ciclomática.

   Pruebas de concurrencia o estrés.

5. **SCRUM es un método ágil general basado en:**

   La administración iterativa del desarrollo de un producto software.

   Los enfoques técnicos de la ingeniería de software.

   Las dos opciones.

6. **El cálculo de complejidad ciclomática V(G) es:**

   V(G)=E-N+2 donde E=aristas, N=Nodos.

   V(G)=E-N-2 donde E=aristas, N=Nodos.

   V(G)=E-N+4 donde E=aristas, N=Nodos.

7. **Una de las técnicas de las pruebas de caja negra es:**

   Partición de clases de equivalencia.

   Pruebas de sistema.

   Pruebas de integración.

8. **La prueba de caja blanca se conoce también como de comportamiento.**

   Cierto.

   Falso.

   Ninguna de las opciones.

9. **Uno de los objetivos de las pruebas de caja negra es:**

   Encontrar errores en funciones incorrectas o faltantes.

   Encontrar errores en estructuras de datos.

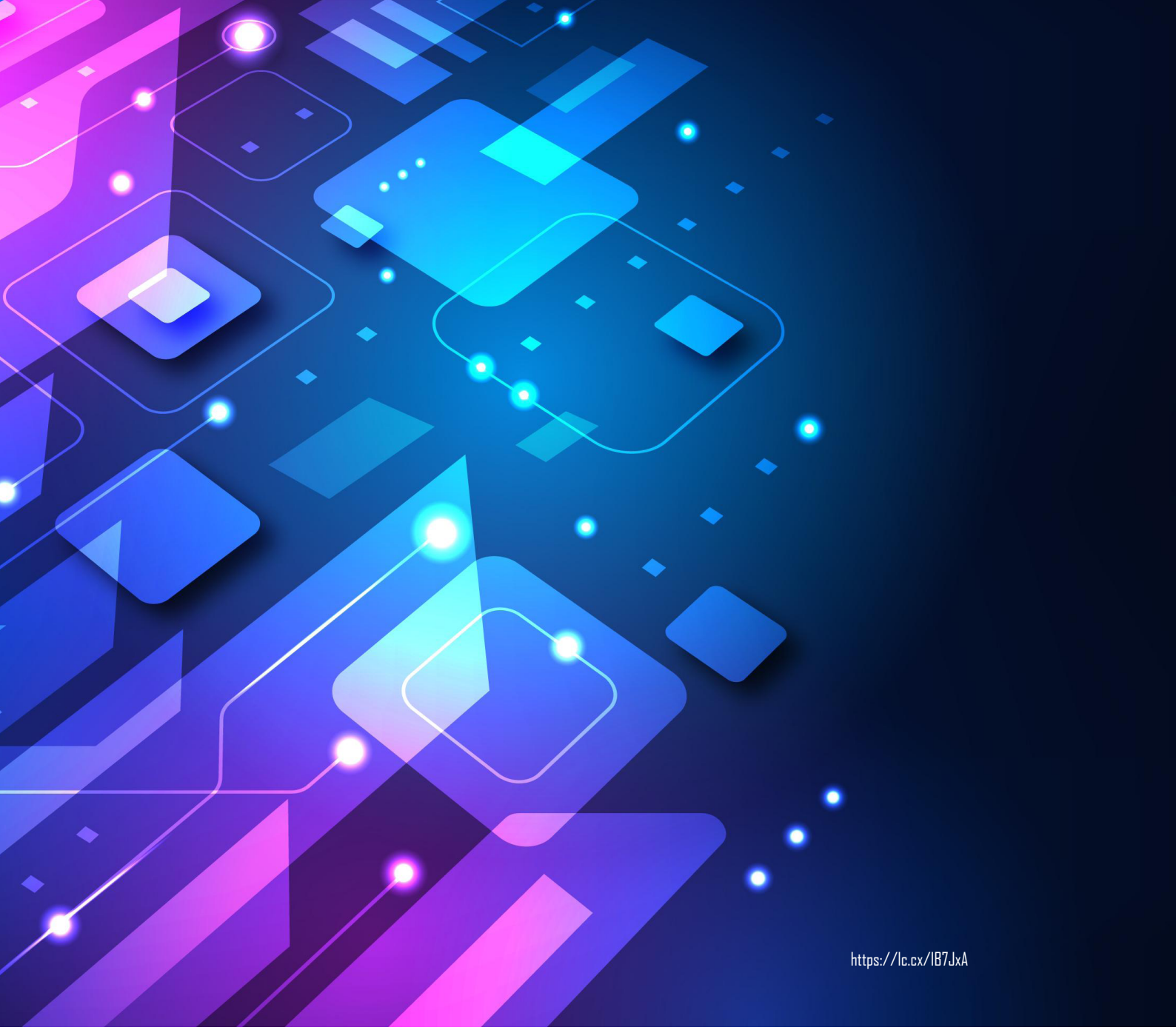
Las dos opciones.

10. La prueba de partición de equivalencia consiste en:

Dividir el dominio de entrada de un programa en clases de datos de los que pueden resultar casos de pruebas.

Dividir el dominio de salida de un programa en clases de datos de los que pueden resultar casos de pruebas.

Dividir el dominio de entrada/salida de un programa en clases de datos de los que pueden resultar casos de pruebas.



CAPÍTULO XII

Marco de trabajo de SCRUM planificación

Primeros pasos

La primera pregunta que seguramente se hará usted como estudiante es: ¿Qué es SCRUM? La respuesta es sencilla. Es un marco de trabajo ágil para desarrollar productos y servicios innovadores de software. Hoy en día, Scrum es un estándar de la industria, y tanto pequeñas como grandes empresas lo están adoptando como método de trabajo (Solis, 2024).

¿Dónde se aplica? Por ser una metodología ágil, Scrum tiene una aplicación de amplio espectro, como se indicó anteriormente. Puede aplicarse al desarrollo de productos y servicios de software de cualquier tamaño, como es el caso de los sistemas ERP para grandes corporaciones o para el desarrollo de pequeños emprendimientos o Startups, ya que aprovecha al máximo los recursos con una mínima inversión (Solis, 2024).

¿Por qué usar SCRUM? Porque es una metodología ágil que brinda eficiencia y obtiene lo mejor de los miembros de un equipo de desarrollo. La adaptación a los cambios los aborda desde la perspectiva de la mejora continua. En este sentido, la adaptación a los cambios imprevistos en el desarrollo de productos software es vital, sobre todo debido a la alta volatilidad de los requisitos funcionales que el equipo de desarrollo tiene que enfrentar en el proceso de desarrollo de un producto software (Solis, 2024).

En la Figura 47, el ciclo de vida en cascada mantiene actividades secuenciales que parten desde la definición de requisitos hasta la operación y el mantenimiento para llegar a entregar un producto software funcional.

Por otro lado, en la Figura 48, el modelo Scrum como marco de trabajo ágil, inicia en las historias de usuario, importantes en la definición de los requisitos funcionales, y de manera casi inmediata pasan a un diseño rápido, por ejemplo, los casos de uso, que son una descripción gráfica de los requisitos y permiten realizar la construcción de código fuente para posteriormente generar el código ejecutable y realizar pruebas de unidad que validen estos requisitos (Pressman, 2005).

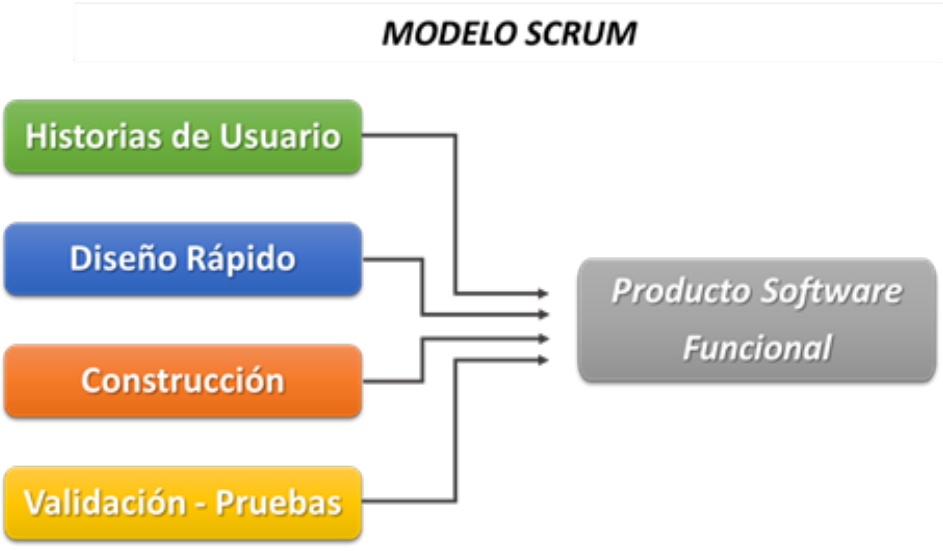
Figura 47

El modelo en cascada o Waterfall

Nota. Modelos de desarrollo. Adaptado de Ciclo de Vida en Cascada https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Figura 48

El modelo Scrum como un sprint



Nota. SCRUM. Adaptado de desarrollo ágil https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Los roles

La Figura 49 presenta a los miembros participantes en el proceso Scrum, cada uno de ellos cumple con un rol y con una responsabilidad; así pues, se tiene a: Los interesados, que son las personas encargadas de asesorar y guiar al dueño del producto en lo referente al problema que se desea resolver. El Scrum master, que prioriza y coordina las actividades a ser desarrolladas, forma parte del equipo y es quien sirve como nexo de comunicación con el dueño del producto. Y finalmente, el equipo de trabajo conformado por analistas, diseñadores y programadores encargados de construir el producto software (Somerville, 2011).

Figura 49

Roles según del modelo Scrum

Nota. Integrantes, roles y responsabilidades SCRUM. Adaptado de roles de Scrum https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

El ciclo de trabajo

En la Figura 50, se muestra el ciclo Sprint, que es la unidad básica de ejecución de la metodología Scrum. Se considera como una sola unidad planificadora donde se prioriza el trabajo a realizar considerando las particularidades que este puede tener. Como actividades del Sprint se consideran a:

Selección donde se identifican los requisitos funcionales, se los prioriza y ordena.

Desarrollo donde se construye código fuente sobre la base de estos requisitos.

Revisión donde se realizan pruebas unitarias al prototipo funcional con el fin de validarlo.

Valoración es realizada por parte del usuario quien aprueba el requisito si este cumple con lo que el necesita (Somerville, 2011).

Figura 50

El ciclo Sprint del proceso SCRUM

Nota. Sprint de SCRUM. Adaptado de ciclo de trabajo de Scrum https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Como se aprecia en la Figura 50, un proceso Scrum tiene tres fases:

1. Planeación del bosquejo, donde se establecen los objetivos generales y el diseño arquitectónico.
2. Ciclo o Sprint, donde cada uno de estos desarrolla un prototipo funcional para el usuario
3. Cierre del proyecto, donde se completa la documentación y concluye el proyecto

Recursos complementarios 12

Consulta el material para reforzar los conocimientos adquiridos

Accede aquí:

Actividad de aprendizaje 12

En el capítulo 12 se desarrolla sobre la aplicación de la planificación de un ciclo SCRUM

Te invito a revisar los siguientes videos para reforzar tus conocimientos

Autoevaluación del capítulo 12

1. **¿Qué es SCRUM?**

   Es un marco de trabajo estructurado para desarrollar productos y servicios innovadores de software.

   Es un marco de trabajo orientado a objetos puros, para desarrollar productos y servicios innovadores de software.

   Es un marco de trabajo ágil para desarrollar productos y servicios innovadores de software.

2. **¿Dónde se aplica SCRUM?**

   Al desarrollo de productos y servicios.

   Pequeñas y medianas PYMES.

   Las dos opciones.

3. **¿Por qué usar SCRUM?**

   Es una metodología estructurada que brinda eficiencia y obtiene lo mejor del equipo de desarrollo.

   Es una metodología ágil que brinda eficiencia y obtiene lo mejor del equipo de desarrollo.

   Es una metodología orientada a objetos que brinda eficiencia y obtiene lo mejor del equipo de desarrollo.

4. **El orden de las fases en el modelo en cascada o Waterfall es:**

   Definición de requisitos/Integración prueba del sistema/Diseño del

sistema y software/Implementación y prueba de calidad/Operación y mantenimiento.

Definición de requisitos/Implementación y prueba de calidad/ Integración prueba del sistema/Diseño del sistema y software /Operación y mantenimiento.

Definición de requisitos/Diseño del sistema y software/Implementación y prueba de calidad/Integración prueba del sistema/Operación y mantenimiento.

5. **El orden de las fases en el modelo SCRUM es:**

   Historias de usuario/Diseño rápido/Construcción/Validación-pruebas.

   Historias de usuario/Construcción/Diseño rápido/Validación-pruebas.

   Historias de usuario/Validación-pruebas/Diseño rápido/Construcción.

6. **Los roles de SCRUM son:**

   Líder del equipo/Dueño del producto, Scrum master, equipo de trabajo.

   Interesados/Dueño del producto, Scrum master, equipo de trabajo.

   Interesados/Dueño del producto, Scrum master, líder del equipo.

7. **El orden en las fases del ciclo de trabajo SCRUM son:**

   Valoración/Selección/Desarrollo/Revisión.

   Desarrollo/Selección/Revisión/Valoración.

   Selección/Desarrollo/Revisión/Valoración.

8. **Las actividades en la fase de selección en un Sprint son:**

   Identificación de requisitos funcionales.

   Priorización y ordenamiento de los requisitos funcionales.

   Las dos opciones.

9. **Las actividades en la fase de revisión en un Sprint son:**

   Realizar pruebas de sistema al prototipo funcional con el fin de validarlo/Valoración por parte del usuario quien aprueba.

Realizar pruebas unitarias al prototipo funcional con el fin de validarlo/Valoración por parte del equipo de desarrollo quien aprueba.

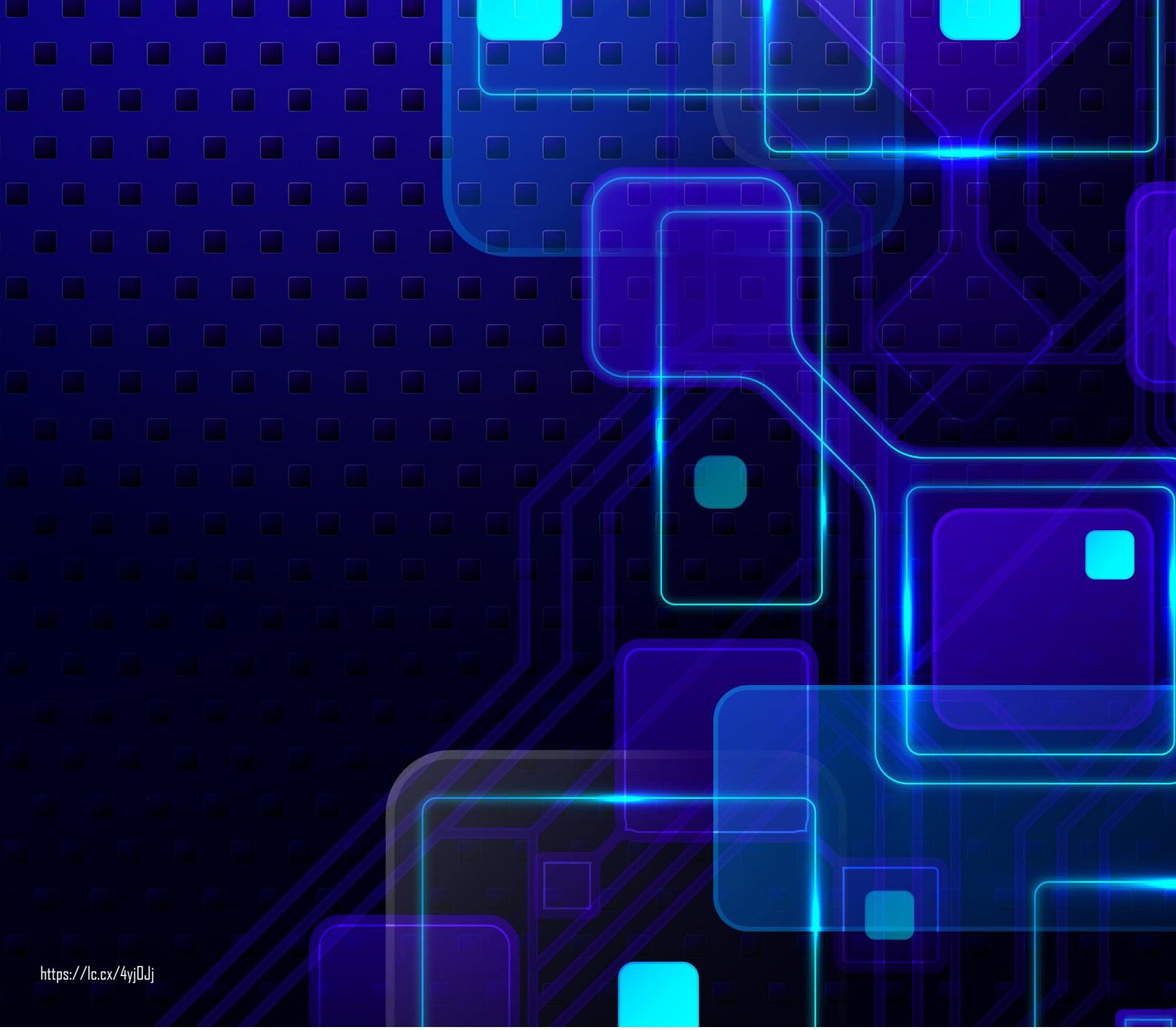
Realizar pruebas unitarias al prototipo funcional con el fin de validarlo/ Valoración por parte del usuario quien aprueba.

10. El equipo de trabajo está conformado por:

Scrum master, diseñadores y programadores encargados de la construcción del producto software.

Analistas, diseñadores y programadores encargados de la construcción del producto software.

Analistas, diseñadores y dueño del producto encargados de la construcción del producto software.



CAPÍTULO XIII

Marco de trabajo de SCRUM validación

Herramientas

Las herramientas diseñadas para empezar con la aplicación del marco de trabajo ágil Scrum se detallan a continuación.

En la Figura 50 se presenta el fundamento del marco de trabajo de historias de usuario (HU). El marco de trabajo para levantar requisitos de software inicia con un primer paso, en el cual, usted como estudiante tiene que responder a siete preguntas (5W+2H) sobre el problema planteado (Ruiz, 2020).

Partiendo de un documento en lenguaje natural como base para la identificación de los requisitos, las preguntas en mención son:

- ¿Qué?
- ¿Por qué?
- ¿Quién?
- ¿Dónde?
- ¿Cuándo?
- ¿Cómo?
- ¿Cuánto?

A cada respuesta del ¿Qué?, que es en realidad el requisito que se busca, se le asigna una prioridad o nivel de importancia que puede ser alta, media o baja y, finalmente, se asigna un orden en la ejecución.

Todas las respuestas, consideraciones de prioridad y secuencia se consolidan en las HU por medio de las cuales usted está en la capacidad de:

1. Explicar el requisito funcional (¿Qué?)
2. Identificar el nombre del requisito (Historia de usuario)
3. Describir brevemente en qué consiste este requisito (¿Cómo?)
4. Definir quién es el responsable (Programador responsable)
5. Delimitar tiempo para ejecutar este requisito (Tiempo) (Ruiz, 2020).

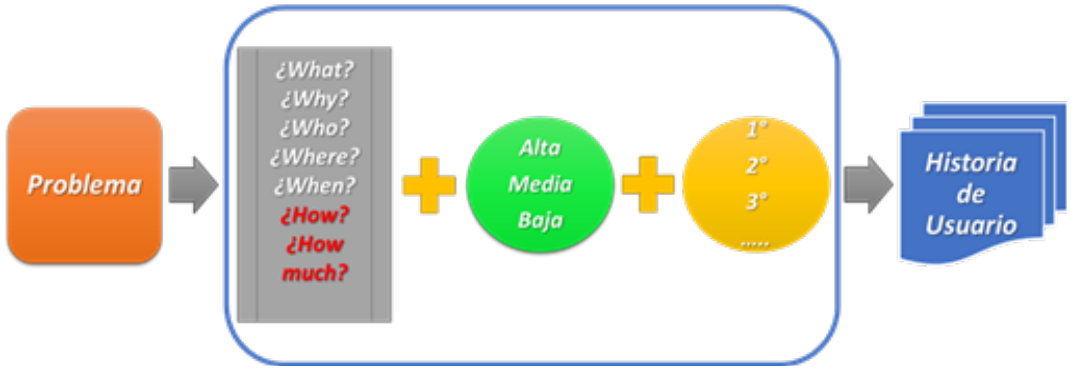
Como un segundo paso, la información obtenida en la HU debe ser revisada por un tutor de quien reciba la retroalimentación que aclare sus dudas. Esta revisión se aplica en varias ocasiones con el objetivo de validar continuamente el proceso de análisis y de síntesis que realiza el estudiante sobre el documento proporcionado por el tutor (Ruiz, 2020).

Como un tercer paso, el tutor revisa la aplicación solicitada, verificando el cumplimiento de requisitos funcionales y la coherencia existente entre ellos, todo esto, previo a la entrega (Ruiz, 2020).

Al final del proceso se cuenta con un conjunto de entregables como las HU diseñadas en MS Excel, el código fuente donde implementaron las HU, el informe final y el trabajo colaborativo del estudiante, donde cuenta sus experiencias en la aplicación de este marco de trabajo para la identificación de requisitos funcionales y la implementación con la metodología Scrum (Ruiz, 2020).

Figura 51

Marco de trabajo de historias de usuario



Nota. Descripción de los componentes del marco de trabajo de historias de usuario. Adaptado de 5W+2H Técnica de análisis de problemas, Progressa Lean, 2014 https://www.progressalean.com/5w2h-tecnica-de-analisis-de-problemas.

En la Figura 52, se indica la matriz resultante de ese proceso de análisis, que a su vez permitirá generar como producto las historias de usuario. La Figura 53 indica un ejemplo donde se detalla los componentes de la HU.

Figura 52

Matriz de marco de trabajo de historias de usuario

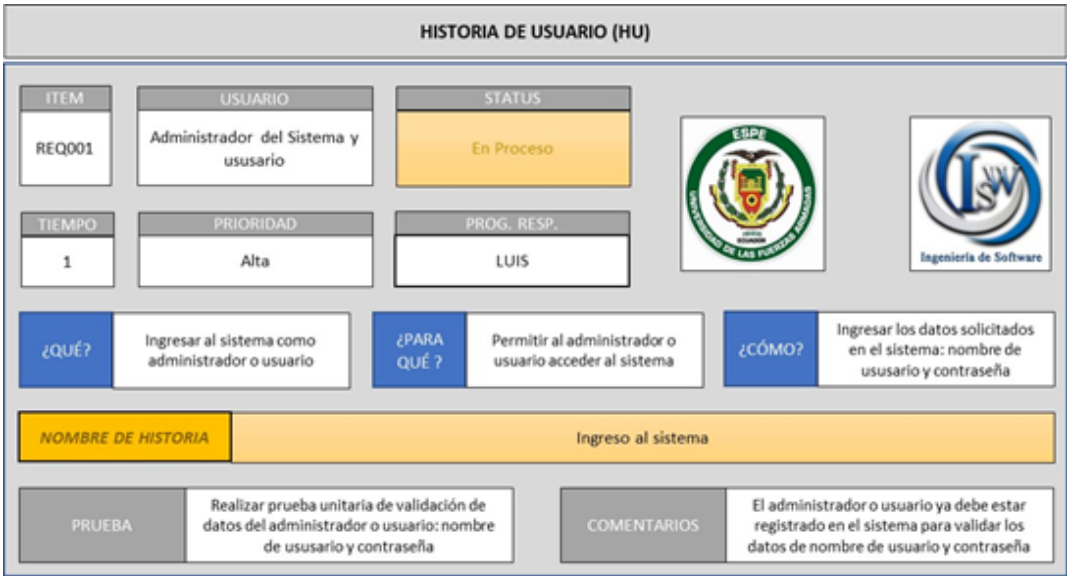
Matriz de Marco de Trabajo de HU "Sistema de gestión de productos y control de un micromercado" / fuente: trabajo académico de G9 NRC3594, periodo 202150

ITEM	PROBLEMA	QUE (NECESIDAD)	PARA QUE (SOLUCIÓN)	PARA QUIEN (USUARIO)	COMO (DESCRIPCIÓN DE TAREAS)	HECHO POR (PROG. RESP.)	CUANTO TIEMPO (ESTIMADO EN HRS)	FECHA DE ENTREGA	PRIORIDAD	STATUS	PRUEBA (COMO SE VERIFICA)	COMENTARIOS	NOMBRE DE HISTORIA
REQ001	El aplicativo debe permitir el acceso del administrador o usuario.	Ingresar al sistema como administrador o usuario.	Permitir al administrador o usuario acceder al sistema.	Administrador del sistema y usuario.	Ingresar los datos solicitados en el sistema: nombre de usuario y contraseña.	luis	1	2021-07-07	Alta	En proceso	Realizar prueba unitaria de validación de datos del administrador o usuario: nombre de usuario y contraseña.	El administrador o usuario ya debe estar registrado en el sistema para validar los datos de nombre de usuario y contraseña.	Ingreso al Sistema
REQ002	El aplicativo debe registrar un nuevo usuario.	Registrar un nuevo usuario al sistema	Almacenar un nuevo usuario en el sistema.	Administrador del Sistema.	Ingresar los datos solicitados en el sistema: nombres, apellidos, cédula, fecha de nacimiento y contraseña.	Magaly	1	2021-07-08	Alta	En proceso	Comprobar en la base datos que el usuario se haya registrado correctamente.	El usuario debe registrar todos los campos solicitados para que se almacenen correctamente en la base de datos.	Registro de Usuario

Nota. Matriz del marco de trabajo de historias de usuario ejemplo de Sistema de gestión de productos y control de inventario de un micro mercado. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Figura 53

Historias de usuario



Nota. Matriz del marco de trabajo de historias de usuario ejemplo de Sistema de gestión de productos y control de inventario de un micro mercado. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Validación y pruebas

La Figura 54 presenta una plantilla con el reporte de errores, que permite llevar un registro de los resultados obtenidos una vez aplicadas las técnicas de caja blanca y de caja negra. Para esta actividad, es recomendable realizar una evaluación por pares, esto es, permitir que un compañero haga las veces de par evaluador y ejecute las pruebas y validaciones necesarias, todo esto con el objetivo de realizar el control de calidad de estos requisitos funcionales, adicionalmente, de esta forma quedan registrados los errores generados en el momento de la prueba (Pressman, 2005).

Figura 54

Reporte de errores de la iteración 1

Reporte de errores e inconsistencias			
Nombre del proyecto	Sistema de gestión de productos y control de inventario de un micromercado		
Fecha de Pruebas:	12/8/2021		
Módulos:	MD1 / MD2		
Analista:	Ing. Jenny Ruiz		
Responsable:	Tavaris; Pillajo; Egas; Achig		
Fecha de Revisión:	13/8/2021		
Identificación Caso prueba	**Descripción de prueba**	**Descripción del error**	**Acciones de corrección**
CP-001/REQ001	Ingreso al sistema	El usuario ingresa datos incorrectos	Mostrar un mensaje indicando el error "datos incorrectos"
CP-002	Interfaz	No cumple ciertos parámetros	Modernizar la interfaz para un mejor manejo e identificación de
CP003/REQ002	Registro de usuario	La cédula no satisface correctamente las pruebas de dígito verificador	Realizar una correción en el código fuente en la validación nde la cédula
CP-004/REQ002	Registro de usuario	Usuario ya existente	Desplegar un mensaje indicando "el usuario ya existe"

Nota. Reporte de errores e inconsistencias de los requisitos números 1 y 2. Adaptado de Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. https://ues-pe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Actividad de aprendizaje 13

El capítulo 13 aplica la planificación de casos de prueba puedes revisar mayor información en el siguiente link

Autoevaluación del capítulo 13

1. **El orden del proceso para obtener historias de usuario es:**

   Secuencia + ¿Qué? ¿Por qué? ¿Quién? ¿Dónde? ¿Cuándo? ¿Cómo? y ¿Cuánto? + Priorización.

   Priorización + ¿Qué? ¿Por qué? ¿Quién? ¿Dónde? ¿Cuándo? ¿Cómo? y ¿Cuánto? + Secuencia.

   ¿Qué? ¿Por qué? ¿Quién? ¿Dónde? ¿Cuándo? ¿Cómo? y ¿Cuánto? + Priorización + Secuencia.

2. **Las historias de usuario le permiten a usted estar en la capacidad de:**

   Explicar el requisito funcional (¿Qué?)/Identificar el nombre del requisito (historia de usuario).

   Describir brevemente en qué consiste este requisito (¿Cómo?)/Definir quién es el responsable (Programador responsable)/Delimitar tiempo para ejecutar este requisito.

   Todas las opciones.

3. **El orden en la ficha de historias de usuario es:**

   Ítem, usuario, estatus, tiempo, prioridad, programador responsable, qué, para qué, cómo, nombre historia, prueba y comentarios.

   Prueba y comentarios, ítem, usuario, estatus, tiempo, prioridad, programador responsable, qué, para qué, cómo, nombre de la historia.

   Usuario, estatus, tiempo, prioridad, programador responsable, qué, para qué, cómo, nombre de la historia, prueba y comentarios, Ítem.

4. **El reporte de errores utilizado en la validación y pruebas es:**

   Lleva el registro de los resultados obtenidos una vez aplicadas las pruebas de estrés.

   Lleva el registro de los resultados obtenidos una vez aplicadas las pruebas de caja blanca y de caja negra.

   Lleva el registro de los resultados obtenidos una vez aplicadas las pruebas de integración y del sistema.

5. **La cabecera del reporte de errores de una iteración tiene:**

   Nombre del proyecto, fecha de prueba, módulo, analista, responsables y fecha de revisión.

   Nombre del caso de prueba, fecha de prueba, módulo, analista, responsables y fecha de revisión.

   Nombre, requisito funcional, fecha de prueba, módulo, analista, responsables y fecha de revisión.

6. **La herramienta llamada marco de trabajo de historias de usuario es fiable y permite obtener requisitos funcionales.**

   Cierto.

   Falso.

   Ninguna de las opciones.

7. **En la matriz de marco de trabajo de historias de usuario, con qué pregunta se identifica el requisito funcional.**

   Qué (necesidad).

   Para qué (solución).

   Cómo (descripción de tareas).

8. **En la matriz de marco de trabajo de historias de usuario, con qué pregunta se identifican las tareas.**

   Fecha de entrega.

   Prioridad/estatus.

   Ninguna de las opciones.

9. **En la matriz de marco de trabajo de historias de usuario, con qué pregunta se identifican las pruebas.**

   Qué es la historia.

   Cuáles son los comentarios.

   Cómo se verifica.

10. **La descripción de los componentes del marco de trabajo de historias de usuario se basa en:**

    Técnica de resolución de problemas de las 5W+2H.

    Técnica de análisis de problemas de las 5W+2H.

    Ninguna de las opciones.

Referencias

Acevedo, W. F. (2014). *Historia y Evolución de la Ingeniería Del Software*. Programa Ingeniería de Sistemas y Telecomunicaciones Asignatura Ingeniería de Software I. https:// chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://ingswusa.wordpress.com/wp-content/uploads/2014/08/historia-de-ing-software.pdf

ACM, (2022). *Computing Curricula.* 2020 [figura]. https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf

Acosta, M. (2016). *Software Enginnering* - Ian Sommerville [figura]. https://es.slideshare.net/MatiasGonzaloAcosta/procesos-de-software-unidad-2-software-enginnering-ian-sommerville

Barajas, M. (2024). *Ingeniería de Software*-Somerville [figura]. https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Barajas, M. (2024). *Ingeniería de Software*-Somerville [tabla]. https://www.academia.edu/41024706/Ingenieria_de_Software_Somerville

Bauer, F. L. (1968). *Software Engineering*. Germany: Nato Science Committee.

Bonder, N. (2023) .21 términos básicos de programación que debes dominar [figura] https://weremote.net/terminos-basicos-programacion/

Charette, R. (2005). *Why Software Fails* [figura]. EEE Spectrum for the Technology Insider: https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf

Estigarribia, R. (2023). Procesos de desarrollo de Software [figura]. https://medium.com/@ramiroec/procesos-de-desarrollo-de-software-d06d18ebd78f

García, F. (2018). *Ingeniería del software*, en Proyecto Docente e Investigador. GRIAL repository: https://repositorio.grial.eu/handle/grial/2451

Hernández, A. (2024). Aplicación del Proceso Unificado de Desarrollo a proyectos de software [figura]. https://www.researchgate.net/profile/Anaisa-Hernandez-Gonzalez/publication/312656269_Aplicacion_del_Proceso_Unificado_de_Desarrollo_a_proyectos_de_software/links/58878a87a6fdcc6b791ec281/Aplicacion-del-Proceso-Unificado-de-Desarrollo-a-proyectos-de-software.pdf

Hinojosa, C. (2014). Repositorio Institucional de la Universidad de las Fuerzas Armadas ESPE. TÉCNICA PARA EL ANÁLISIS DE REQUISITOS DE SOFTWARE BASADA EN LA GESTIÓN DEL CONOCIMIENTO, PARA LA EMPRESA DE DESARROLLO DE SOFTWARE KRUGER: http://repositorio.espe.edu.ec/handle/21000/7720

Hinojosa, C. (2019). *Introducción a la ingeniería del software.* Scribd: https://es.scribd.com/document/462507105/Introduccion-Ingenieria-de-Software

IEEE Computer, S. (2014). *Guide to the Software Engineering Body of Knowledge Version 3.0* (SWEBOK Guide V3.0. https://www.academia.edu/8583868/Guide_to_the_Software_Engineering_Body_of_Knowledge_Version_3_0_SWEBOK_Guide_V3_0

IEEE Xplore. (2002). EEE Standard Glossary of Software Engineering Terminology. he Institute of Electrical and Electronics Engineers, Inc.: https://ieeexplore.ieee.org/document/159342

IEEE. (2014). swebok v3.0 [figura]. https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf

IngSistemas. (2017). ¿Qué es la ingeniería de software? [figura]. https://ing-sistemas.com/2017/02/09/que-es-la-ingenieria-software/

Jacobson, I., & Booch, G. &. (2000). El proceso unificado de desarrollo de software (primera edición). Madrid, España: S.A., Pearson Educación. https://www.academia.edu/11946867/El_Proceso_Unificado_de_desarrollo_de_sofware?sm=b

Ledesma. (2018). Essays Club Español. Este trabajo es hecho con el fin de dar a conocer la historia de la ingeniería de sistema llamada en otros países ingeniería de software e ingeniería de computación: https://es.essays.club/Ciencias-sociales/Historia/Este-trabajo-es-hecho-con-el-fin-de-184216.html

López Echeverry, A. M., & Valencia Ayala, L. E. (2008). *Introducción a la calidad de software*. Dialnet: https://dialnet.unirioja.es/servlet/articulo?codigo=4745899

Modelado del sistema. (2016). Modelos de contexto [figura]. https://modeladodelsistema.wordpress.com/2016/06/15/

Oktaba, H. (2018). 50 años de la ingeniería de software problemas, logros, tendencias y retos. SG#58 tejiendo nuestra red. https://sg.com.mx/revista/58/50-anos-de-la-ingenieria-de-software-problemas-logros-tendencias-y-retos

Ortiz Herrera, M. (2001). e-REdING, repositorio elaborado por la Biblioteca de Ingeniería. MÉTODOS Y TÉCNICAS PARA LA GESTIÓN DE PROYECTOS SOFTWARE: https://biblus.us.es/bibing/proyectos/abreproy/70193/

Perea, A. (2021). capítulo 4 *Ingeniería de Requerimientos* Somerville [figura]. https://slideplayer.es/slide/17988345/

Piattini, M. (2016). Evolución de la ingeniería del software y la formación de profesionales. *Revista Institucional de la Facultad de Informática UNLP*. http://sedici.unlp.edu.ar/bitstream/handle/10915/57358/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Platzi. (2021). Atributos de calidad de un producto de software [figura]. https://platzi.com/tutoriales/1248-pro-arquitectura/5498-atributos-de-calidad-de-un-producto-de-software/

Pohl, K. & Rupp, C. (2015). *Santa Barbara CA: Rocky Nook Inc. Requeriments Engineering Fundamentals: a study guide for the certified professional for requirements engineering exam, foundation level--IREB compliant* (second edition): https://rockynook.com/shop/computing/requirements-engineering-fundamentals-2nd-edition/

Potts, C. (1993). *Software-Engineering Research Revisited*. IEEE Software: https://ieeexplore.ieee.org/abstract/document/232392

Pressman, R. (2001). *Software Engineering A PRACTITIONER'S APPROACH* (Fifth ed.) [figura]. https://repository.dinus.ac.id/docs/ajar/Software_Engineering_-_Pressman.pdf

Pressman, R. (2005). *Un enfoque práctico* (Séptima ed.) [figura]. https://docs.google.com/document/d/1x1uFkX13aWHfVkssPTqSZgRDN-b0GiY-VXuSScnQ3YCs/edit?usp=sharing

Pressman, R. (2005). *Un enfoque práctico* (Séptima ed.) [tabla]. https://docs.google.com/document/d/1x1uFkX13aWHfVkssPTqSZgRDN-b0GiY-VXuSScnQ3YCs/edit?usp=sharing

Pressman, R. 7. (2005). (M. Hill, Editor) https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf

PROGRESSA LEAN. (2014). 5W+2H Técnica de análisis de problemas [figura]. https://www.progressalean.com/5w2h-tecnica-de-analisis-de-problemas

Real Academia Española. (2020). *Diccionario de la lengua española*. https://dle.rae.es/requisito?m=form

Romero, J. (2017). Gestión de Pruebas y Visualización para Aplicaciones que Efectúan Tratamiento de Imágenes Digitales [figura]. https://www.researchgate.net/figure/Figura-1-Estrategias-de-Prueba-Pressman-2001_fig1_318457069

Ruiz, J. (2020). Planteamiento de un marco de trabajo para levantar requisitos de software con estudiantes del primer nivel de la carrera de software de la Universidad de las Fuerzas Armadas ESPE. Sangolqui, Ecuador.

Ruiz, J. (6 de Mayo del 2024). *Ingeniería de Software*.

Skrobak, G. (2023). Proceso Unificado de Desarrollo de Software [figura]. Phttps://www.academia.edu/11946867/El_Proceso_Unificado_de_desarrollo_de_sofware?email_work_card=view-paper

Solis, C. (2024). In Learning. Scrum esencial: https://www.linkedin.com/learning/aprende-scrum/presentacion-del-curso-aprende-scrum

Somerville, I. (2011). Software Engineering 9. https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/index.html

Tavaris, L. Egas, D. Pillajo, M. y Achig E, 2021. Partición de clases equivalentes [tabla]. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Tavaris, L. Egas, D. Pillajo, M. y Achig E.. (2021). Prueba de caja blanca "Sistema de gestión productos y control de inventario de un micro mercado". [figura]. https://uespe-my.sharepoint.com/:f:/g/personal/jaruiz_espe_edu_ec/Ev4o4sbzDFxDnjGsUrdzytcBaOFO7ZVPz8X_x5_mU_NtwA?e=0Ghu7O

Testing IT. (2022). Pruebas para atributos de calidad [figura]. https://www.testingit.com.mx/blog/atributos-de-calidad-de-software

Tesuva. (2014). *Ingenieria del Software. Un Enfoque Practico* - tesuva.edu.co. Tesuva.edu.co: https://tesuva.edu.co/phocadownload/Ingenieria_del_Software._Un_Enfoque_Practico.pdf

Young, R. (2004). The Requirements Engineering. Boston London: Artech House, INC. The Requirements Engineering: https://www.acqnotes.com/Attachments/The%20Requirements%20Engineering%20Handbook%20by%20Ralph%20R.%20Young.pdf

Zap, C. A. (2024). ZAPTEST. Pruebas de Caja Blanca: ¡Qué es, Cómo funciona, Retos, Métricas, Herramientas y Más: https://www.zaptest.com/es/pruebas-de-caja-blanca-que-es-com-funciona-retos-metricas-herramientas-y-mas

Fundamentos de ingeniería de software
Un enfoque práctico

"Fundamentos de Ingeniería de Software: un enfoque práctico" presenta una exposición clara, rigurosa y estructurada de los principios esenciales de la Ingeniería de Software (IS).

Dirigido a estudiantes y docentes universitarios, este texto proporciona un enfoque eminentemente práctico para el desarrollo de sistemas de alta calidad. A través de sus capítulos, se abordan los temas más relevantes de la IS, desde sus antecedentes históricos hasta las metodologías ágiles y tradicionales, como RUP y SCRUM, integrando conceptos fundamentales, técnicas de análisis, diseño, validación de requisitos y gestión de la calidad.

Cada sección incorpora actividades de aprendizaje y ejercicios de autoevaluación que refuerzan la comprensión activa del lector. Con un estilo técnico y accesible. Esta obra constituye una referencia indispensable para dominar de manera integral el proceso de desarrollo de software, equilibrando teoría y práctica en una propuesta académica sólida y actualizada.

ISBN: 978-9942-652-26-3

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA