UTPL
La Universidad Católica de Loja

Vicerrectorado de Modalidad Abierta y a Distancia

Fundamentos de Ingeniería de Software

Guía didáctica

UTPL

Facultad Ingenierías y Arquitectura

Fundamentos de Ingeniería de Software

Guía didáctica

Carrera	PAO Nivel
Tecnologías de la información	V

Autores:

Marco Patricio Abad Espinoza

Reestructurada por:

Danilo Rubén Jaramillo Hurtado

SIST_3012

Universidad Técnica Particular de Loja

Fundamentos de Ingeniería de Software

Guía didáctica

Marco Patricio Abad Espinoza
Reestructurada por:
Danilo Rubén Jaramillo Hurtado

Diagramación y diseño digital

Ediloja Cía. Ltda.
Marcelino Champagnat s/n y París
edilojacialtda@ediloja.com.ec
www.ediloja.com.ec

ISBN digital -978-9942-25-770-3

Año de edición: abril 2020

Edición: primera edición reestructurada en septiembre 2024 (con un cambio del 5%)

Loja-Ecuador

partir del material citando la fuente, bajo los siguientes términos: ***Reconocimiento****- debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios.* Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante. *No Comercial-no puede hacer uso del material con propósitos comerciales. Compartir igual-Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.* No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia. https://creativecommons.org/licenses/by-nc-sa/4.0/

Índice

1. Datos de información

1.1 Presentación de la asignatura

1.2 Competencias genéricas de la UTPL

Trabajo en equipo.

1.3 Competencias específicas de la carrera

- Diseñar aplicaciones de *software* que permitan mediante técnicas avanzadas de modelado dar solución a los requerimientos del cliente utilizando estándares de la industria.
- Asegurar la calidad tanto de los productos como de los procesos en los proyectos informáticos, utilizando buenas prácticas definidas por la industria para garantizar sistemas eficientes y negocios rentables.
- Gestionar la implementación de soluciones de negocio mediante la ejecución de proyectos de TI que cumplan adecuadamente los requisitos especificados por la organización.

1.4 Problemática que aborda la asignatura

Esta asignatura aborda la problemática de la debilidad en el tejido empresarial, enseñando métodos, técnicas y herramientas para crear aplicaciones de *software* para empresas, ayudando a las empresas a ser más eficientes y a incrementar su capacidad operativas, ello mejora la situación socioeconómica de la región y del país, haciéndolos más productivos y creando fuentes de empleo.

2. Metodología de aprendizaje

La metodología de trabajo corresponde al Aprendizaje Basado en Proyectos, el cual permite a los estudiantes a más de aprender los conceptos y metodologías, aplicarlos en la creación de una solución a partir de una problemática, a esta problemática ellos la analizan, buscan estrategias de solución y proponen un proyecto que genere la solución desde distintos puntos de vista. Los estudiantes además contarán con un laboratorio virtual donde desarrollarán sus prácticas y experimentación.

3. Orientaciones didácticas por resultados de aprendizaje

Primer bimestre

Resultado de aprendizaje 1:

Conoce las principales áreas de conocimiento de la Ingeniería de Software.

Para alcanzar el resultado de aprendizaje 1, se introducirá a los estudiantes en los fundamentos de la Ingeniería de *Software*. Se estudiará el origen de la disciplina, sus conceptos fundamentales y las principales actividades involucradas en el desarrollo de *software*. Además, se abordará el desarrollo profesional en el campo, proporcionando una comprensión completa de cómo se organiza y gestiona el trabajo en esta área. Esto permitirá a los estudiantes aplicar eficazmente estos conocimientos en proyectos reales y en su futura carrera profesional.

Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.

Semana 1

El desarrollo de *software* es uno de los ejes centrales de formación del ingeniero en tecnologías de la información, a lo largo de la carrera usted se ha formado en competencias de programación y de modelado de sistemas, sin embargo, para desarrollar *software* desde una perspectiva profesional,

hacen falta otras habilidades como la de analizar un problema de negocio y proponer soluciones que incluyan *software*, identificar las necesidades del cliente y transformarlas en un diseño antes de programar (construir) y hacer todo esto adoptando metodología que garantice un correcto trabajo en equipo donde diferentes personas trabajan en equipo para construir la solución con los niveles de calidad requeridos en un entorno de negocio. Este conjunto de habilidades, métodos y herramientas es lo que estudia la Ingeniería de *software*, y en esta asignatura estudiaremos sus fundamentos, los cuales se complementarán a futuro con otras asignaturas.

Unidad 1. Introducción a la ingeniería de software

Esta unidad es introductoria y trataremos de poner en contexto los elementos y conceptos importantes de la ingeniería de *software*, y sobre todo trataremos de demostrar la importancia que tiene en la carrera y en el desarrollo profesional.

Para el estudio de esta unidad utilizaremos los siguientes recursos educativos:

- **Texto básico**: Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación, el curso OCW; comprende los temas generales de la sobre ingeniería de *software*.
- Guía didáctica: Abad, M. (2020). Guía didáctica Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL, en la que se desarrolla la mayoría de los contenidos de la asignatura, en este apartado utilizará la unidad 1.
- Materiales educativos:
    - Hernández, J. Ruiz, F. (2011). Ingeniería de *software* Cantabria. Donde obtendrá mucha información sobre varios de los temas abordados en el presente curso.
    - Peñalvo, F. J. G., Martin, S. B., González, M. A. C. (2008,). Ingeniería del *software*. Retrieved March 27, 2020, from OCW-USAL.

1.1. Origen de la ingeniería de software

Para comprender la importancia de la Ingeniería de *software* para las ciencias de la computación, nos remontaremos a finales de 1950, el cual se considera un período esencial en la era de la computación, según (Wirth, 2008), fue la época en la que los grandes ordenadores se pusieron a disposición de centros de investigación y universidades, y fueron utilizados principalmente en ingeniería y ciencias naturales, en esa época también ganaron importancia para su uso en empresas, y nació la profesión de los programadores, la cual era considerada una actividad sofisticada que era desarrollada por gurús de las computadoras.

A inicios de los 60 se crearon los primeros lenguajes de programación, el primero de ellos fue Fortran, que era un lenguaje orientado a trabajar con fórmulas matemáticas, y que fue desarrollado por IBM en 1957, luego apareció su sucesor denominado ALGOL en 1958. El primer lenguaje orientado al desarrollo de aplicaciones para negocios fue COBOL, desarrollado en 1962 por el Departamento de Defensa de los Estados Unidos. Esos sistemas procesaban sus trabajos por lotes (*batch processing*), eso significaba que todos los trabajos se recolectaban antes de iniciar el procesamiento y tenían que colocarse en una cola de trabajos, no se empezaba a cargar otro trabajo hasta que finalice el primero, y puesto que los dispositivos de entrada como las cintas y tarjetas perforadas eran lentos, había un gran desperdicio del tiempo de procesador haciéndolo muy costoso.

En 1963, surgió el primer sistema de tiempo compartido, diseñado por John McCarthy en el MIT (Massachusetts Institute of Technology) y fue implementado en un ordenador DEC PDP-1, este sistema permitía múltiples usuarios trabajando al mismo tiempo, haciendo necesario guardar el estado de cada trabajo en la máquina. Dada su eficiencia, IBM y General Electric fabricaron sistemas de tiempo compartido para sus *mainframes* y como consecuencia la complejidad se incrementó considerablemente.

Las principales dificultades encontradas fueron que los sistemas se anunciaron, pero no podían entregarse a tiempo, los problemas de operación se hicieron muy difíciles de resolver. El multiprocesamiento y la programación concurrente eran los ingredientes centrales de los sistemas de tiempo compartido y los programadores no contaban con la experiencia necesaria para trabajar en este tipo de aplicaciones, en consecuencia, se prometieron sistemas que no pudieron entregarse a tiempo y esto afectó a muchas compañías grandes y debido a ello estuvieron al borde del colapso.

Con este antecedente, en 1968 la OTAN (Organización del Tratado del Atlántico Norte), patrocinó una conferencia (*Software Engineering Reports*) en la cual se discutieron abiertamente las dificultades de la programación y se determinó que las técnicas de programación utilizadas en la época eran inadecuadas y que debían adoptarse nuevos métodos. En este contexto fueron acuñados los términos ingeniería de *software* y crisis del *software*. Lo cual dio origen a nuevos esfuerzos por resolver la complejidad y hacer que la programación pase de ser un arte a una disciplina de ingeniería.

Ahora revise la lectura correspondiente, el tema 1 del OCW Ingeniería de *software* I. Vaya a materiales de clase y descargue el documento MC-F-002 y estudie el apartado Evolución histórica del desarrollo de *software*.

Como habrá notado, la ingeniería de *software* es una ciencia relativamente nueva y debido a la evolución de la tecnología cada vez tiene más desafíos, y los niveles de complejidad son cada vez más altos.

En razón de ello, vale la pena plantearse las siguientes interrogantes:

¿Cuáles son los factores que inciden en la complejidad del *software*?, ¿qué cambios importantes en los métodos, técnicas y herramientas de ingeniería de *software* se han producido en los últimos años para atender los problemas que enfrentan los proyectos de desarrollo de *software*?, ¿cuáles son las tendencias en el campo de la ingeniería de *software*?

1.2. Conceptos de ingeniería de software

En el apartado anterior, hemos realizado un breve recorrido por los orígenes de la ingeniería de *software*, el cual nos ha servido para plantearnos el porqué de su importancia, ahora vamos a revisar algunos conceptos importantes.

Comience revisando la tabla 1.1 del **texto básico**, en la página 6, donde se plantean algunos conceptos importantes en este campo.

Para complementar esta información, tome nuevamente el documento del tema 1 del recurso OCW de Ingeniería de *software* 1, trabajado en el apartado anterior, revise los apartados "Problemática del desarrollo de *software*" y "Contexto de la ingeniería de *software*".

Una vez estudiados, debería ser capaz de describir con precisión los siguientes interrogantes:

- ¿Qué es el *software*?
- ¿Cuáles son los atributos de un buen *software*?
- ¿Cuáles son las actividades de la ingeniería de *software*?
- ¿Qué es una ingeniería?
- ¿Qué es y qué hace un ingeniero?
- ¿Qué es un sistema?
- ¿Qué es un proceso?
- ¿Qué es un proyecto?
- ¿Qué es un usuario y qué tipos de usuarios hay?

Con seguridad logró responder a cada una de las interrogantes planteadas, sin embargo, en ellas no se ha mencionado lo que es la ingeniería de *software*, por lo tanto, vamos a revisar otros conceptos importantes:

1.2.1 Ingeniería de software

El glosario estándar de ingeniería de *software* del Instituto de Ingenieros Eléctricos y Electrónico (IEEE, por sus siglas en inglés) (*Institute of Electrical and Electronics Engineers*, 1990), define a la ingeniería de *software* como:

1. La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del *software*, esto es la aplicación de ingeniería al *software*.
2. El estudio de enfoques como en 1).

Otra definición dada en (Kung, 2014) dice:

> "Ingeniería de *software* como disciplina es enfocada en la investigación, educación y aplicación de procesos de ingeniería y métodos para incrementar significativamente la productividad y la calidad del *software* mientras se reducen los costos del *software* y el tiempo de comercialización".

Si analizamos estas dos definiciones, podemos establecer que la ingeniería de *software*, rompe el esquema de la simple codificación de programas, y pasa al "estudio y aplicación de ingeniería al diseño, desarrollo y mantenimiento de productos de *software*" (Ahmed & Bhanu, 2016), y para entender lo que significa aplicar ingeniería a los procesos de desarrollo de *software*, vamos a remitirnos a los aspectos que plantean (Bruegge & Dutoit, 2010), respecto de lo que es la ingeniería de *software*:

- Es una actividad de **modelado,** lo cual permite a los ingenieros simplificar problemas complejos enfocándose en una parte de la realidad, los ingenieros pueden desarrollar varios modelos del sistema y del dominio de la aplicación.
- Es una actividad de **resolución de problemas**, y para ello usan los modelos que les permitan llegar a soluciones aceptables, para ello puede realizar actividades de experimentación y debido a las limitaciones en cuanto a recursos y tiempo suelen utilizar técnicas

empíricas para evaluar las ventajas y desventajas de las diferentes alternativas de solución.

- Es una actividad de **adquisición de conocimiento.** Cuando se modela la aplicación, los ingenieros de *software* recolectan datos, los organizan en información y la formalizan en conocimiento.
- Es una actividad **fundamentada en la razón.** Cuando se adquiere conocimiento y se toma decisiones acerca del sistema o de su dominio de aplicación, los ingenieros también necesitan capturar el contexto en el cual se tomaron esas decisiones y el fundamento detrás de las mismas, este fundamento debe permitir a los ingenieros comprender las implicaciones de los cambios propuestos cuando se revisa estas decisiones.

A continuación, en el siguiente módulo didáctico se analiza el significado y las implicaciones de cada uno de estos aspectos:

Aspectos de la ingeniería de software

1.2.2 Participantes y roles

Un proyecto de *software* requiere el trabajo en equipo de muchas personas, cada una de las cuales tiene diferentes habilidades e intereses en el proyecto, como ejemplo podemos mencionar el cliente, los usuarios, el gestor del proyecto, los desarrolladores y escritores técnicos. Decimos, por tanto, que cada uno de estos asume un rol en el proyecto y cada rol tiene sus propias responsabilidades, puede haber varias personas ejecutando el mismo rol y un participante también puede ejecutar más de un rol.

Dependiendo del proyecto y del proceso de desarrollo adoptado para el proyecto, las personas pueden ejecutar diferentes roles. En la tabla 1, se describen algunos de los roles comunes en proyectos de desarrollo de *software*, así como sus responsabilidades.

Tabla 1

Roles comunes en proyectos de desarrollo de software

Rol	Responsabilidades	Ejemplo
Cliente	• Define el problema o necesidad del negocio. • Proporciona los requerimientos de alto nivel para la definición del alcance. • Proporciona los criterios de éxito del proyecto.	Una empresa o su representante. Un gerente de un departamento.
Usuario	• Provee el conocimiento respecto de las actividades que debe realizar en sus actividades diarias y que servirán para establecer los requerimientos del sistema.	Un vendedor. Un cajero de un banco. Una secretaria.
Gestor	• Responsable de la organización del trabajo del equipo. • Contrata personas, asigna tareas, monitorea el avance del proyecto, gestiona o provee entrenamiento al equipo del proyecto, administra los recursos asignados por el cliente. • Responsable de la entrega de los resultados y la culminación del proyecto.	Gestor del proyecto. Scrum Máster
Desarrollador	• Responsable de las actividades de especificación, diseño, implementación y pruebas del sistema.	Analista. Programador. Tester.
Escritor técnico	• Escribir la documentación que será entregada al cliente. • Entrevista a los desarrolladores, gestores y usuarios para comprender el sistema.	Documentador. Elaborador de manuales de usuario.

Nota. Abad, M., 2020

1.2.3 Productos de trabajo

Para (Bruegge & Dutoit, 2010), los productos de trabajo son artefactos (elementos tangibles) que se producen durante el proceso de desarrollo, y estos pueden ser para consumo interno del equipo de proyecto en cuyo caso adoptan el nombre de **productos de trabajo internos,** y existen productos de trabajo que deben entregarse al cliente, a estos productos de trabajo se los conoce como **entregables,** los entregables se definen en el *contrato* y forman parte del alcance del proyecto, estos son los resultados visibles que agregan valor al cliente.

Tenga en cuenta que el *contrato* es el documento formal que firma el cliente con la empresa desarrolladora, donde se listan los productos que se entregarán, las condiciones de entrega y las acciones a tomar en caso de incumplimiento.

Los productos de trabajo interno pueden clasificarse en documentación técnica, reportes de avance, modelos, código fuente, por ejemplo, en documentación técnica se puede tener: Normas de programación, manuales de pruebas; en lo referente a modelos puede tener: modelos de *software*, modelos de datos. En cuanto a los entregables se puede tener categorías como: Documentación de usuario, programas ejecutables, documentos de aprobación, por ejemplo, documentación se puede tener manuales de usuario, manual de configuración; programas ejecutables: módulos de la aplicación, librerías.

1.2.4 Requerimientos

Los **requerimientos de usuario** son “descripciones de las propiedades necesarias y suficientes de un producto que satisfará las necesidades del consumidor” y **los requerimientos de** *software* son “descripciones de las

propiedades necesarias y suficientes del *software* que deben cumplirse para asegurar que el producto logre el objetivo para lo que fue diseñado" (Gottesdiener, 2005).

Por tanto, los requerimientos de *software* son las especificaciones de las características que el producto de *software* debe satisfacer con el fin de lograr la aceptación de los usuarios. Aunque este tema se desarrollará con mayor detalle más adelante, es importante mencionar que los requerimientos de *software* pueden ser funcionales y no funcionales.

Los **requerimientos funcionales** describen las funciones que el sistema debe permitir en relación con las actividades del usuario para cumplir con sus actividades de negocio, y los **requerimientos no funcionales** especifican restricciones de operación del sistema.

Para ilustrar estos requerimientos, piense en un sistema para operar una biblioteca en una universidad, en donde se ha identificado los requerimientos descritos en la tabla 2 analícelos y establezca diferencias entre ellos.

Nota: los requerimientos no funcionales se asocian a la calidad del producto, concretamente a ciertas características denominadas atributos de calidad, y tienen un alto impacto en las decisiones de diseño, por lo tanto, es muy importante que no pase de este apartado si no logra completar exitosamente la actividad 1.

Recuerde que en todo momento puede acudir a su tutor para aclarar cualquier duda que se le presente durante su proceso de aprendizaje.

Tabla 2

Ejemplo de requerimientos funcionales y no funcionales para un sistema de biblioteca.

ID	Requerimiento
Requerimientos funcionales	
F01	El sistema debe permitir crear una ficha de datos de los libros que incluya: título, autor (es), año de edición, editorial, tabla de contenidos, descriptores, materia.
F02	El sistema debe permitir registrar una ubicación física de los libros.
F03	Debe permitir la clasificación del libro de acuerdo al sistema decimal Dewey.
F04	Debe permitir la creación de un código único del libro, para identificarlo en los diferentes procesos operativos.
F05	El sistema permitirá la impresión de etiquetas identificativas que incluyan, el título, el apellido del autor, el año de publicación, la ubicación física y el código único del libro.
F06	El sistema debe permitir la búsqueda de los libros por autor, materia, descriptor, título, editorial.
F07	El sistema debe permitir el registro de préstamos a estudiantes y docentes de la Universidad.
F08	El sistema debe permitir el registro de las devoluciones y el reingreso del libro a circulación.
Requerimientos no funcionales	
N01	El sistema debe brindar protección de acceso con contraseña a los componentes de registro de material, préstamos y devoluciones sólo a los usuarios autorizados.
N02	El tiempo máximo que debe demorar la entrega de un libro en préstamo es de 1 minuto.
N03	El módulo de consultas debe ser una aplicación web.
N04	El sistema debe enviar notificaciones al usuario un día antes del plazo de devolución de los libros prestados.

Nota. Abad, M., 2020.

Continuemos con el aprendizaje mediante la revisión de las actividades y desarrollo profesional en el software.

1.3. Actividades de desarrollo de software

Como se mencionó en el apartado anterior, la ingeniería de *software* implica mucho más que la programación de aplicaciones, en este apartado vamos a revisar las actividades genéricas que se realizan durante el desarrollo de *software*, independientemente de la metodología o el enfoque que se utilice.

Antes de comenzar, es necesario hacer algunas precisiones respecto de los términos actividad, tarea y recursos.

Una actividad "Es un proceso que tiene lugar en el tiempo y en el espacio, y en el cual un agente actúa con unos objetivos determinados" (Sánchez, Sicilia, & Rodríguez, 2012), las actividades, por tanto, son un conjunto de tareas que se ejecutan con el fin de obtener un artefacto, y pueden ser actividades generales en cuyo caso incluyen otras actividades específicas. Las actividades se componen de tareas y pueden ser asignadas a un **recurso**. Una **tarea** es una unidad mínima de trabajo que puede ser gestionada, e implica la ejecución de los diferentes pasos que una persona debe ejecutar para completar las actividades.

Una actividad es asignada a un recurso (desarrollador), quien realiza las tareas necesarias para cumplir con la actividad, las tareas consumen recursos y producen entregables. Los **recursos** pueden ser suministros para desarrollar las tareas, tiempo y equipamiento.

En los procesos de planificación un gestor descompone el trabajo, en actividades y en tareas a las cuales les asigna recursos. Estos conceptos se entienden mejor con los ejemplos de la tabla 3.

Tabla 3

Ejemplos de actividades, tareas y recursos

Ejemplo	Tipo	Descripción
Levantamiento de requerimientos.	Actividad	Permite obtener el artefacto Especificación de requerimientos, e incluye las tareas necesarias para obtener y validar los requerimientos del lado cliente.
Desarrolla el caso de prueba "Sin valor exacto" para el sistema de cajero automático.	Tarea	La tarea asignada al recurso Jhon (tester) se enfoca en verificar el comportamiento del cajero automático cuando se intenta retirar dinero y este no tiene las denominaciones de billete exactas para entregar el valor solicitado por el cliente. Incluye especificar el entorno de las pruebas, la secuencia de entradas a probar y las salidas esperadas.
Base de datos de cuentas.	Recurso	Incluye un ejemplo de la estructura de la base de datos de cuentas para poder realizar pruebas.

Nota. Adaptado de Object-Oriented Software Engineering, por Bruegge, B., & Dutoit, A. H., 2010, USA: Prentice Hall.

Dependiendo de los autores, las actividades de desarrollo de *software* se plantearon originalmente como análisis, diseño implementación, pruebas y mantenimiento, pero esta percepción ha ido evolucionando con el tiempo y hoy en día se cuenta con diferentes enfoques y metodologías de desarrollo de las que nos ocuparemos en la siguiente unidad.

Para estudiar este tema, remítase a la guía didáctica y, estudie el apartado 2.2 Actividades del proceso. En él se explica una versión genérica de las actividades del proceso de desarrollo y le ayudarán a comprender en qué consiste el desarrollo de *software*.

Para complementar el tema, y puesto que la presente asignatura se centra en Ingeniería de *software* orientada a objetos, a continuación, se describen las actividades que se desarrollan bajo este paradigma de programación, que en sí no constituye una metodología, sino que describe los procesos necesarios para construir *software*:

- **Levantamiento de requerimientos. –** Los clientes y desarrolladores establecen el propósito del sistema y lo describen en términos de actores y **casos de uso**. Los actores son los usuarios o dispositivos que interactúan con el sistema, y los casos de uso son descripciones de las posibles acciones que el actor puede realizar en el sistema a través de una funcionalidad.
- **Análisis. –** Los desarrolladores producen un modelo del sistema, transformando los casos de uso en un modelo de objetos consistente que facilite el diseño del mismo, se elimina ambigüedades y mejora la comprensión de las características del sistema. El modelo de análisis incluye clases junto con sus atributos, operaciones y asociaciones.
- **Diseño del sistema. –** Los desarrolladores establecen las metas del diseño del sistema y lo descompone en subsistemas más pequeños, además se establecen las estrategias para construir el sistema, se incluye plataformas de *hardware* y *software*, la persistencia de los datos, el flujo de control global, las políticas de acceso y el manejo de las condiciones operativas y de mantenimiento del sistema.
- **Diseño de objetos. –** Los desarrolladores definen los objetos del dominio de la solución para reducir la brecha entre el modelo de análisis y la plataforma de implementación definida durante el diseño. Se describen, por tanto, objetos e interfaces de subsistemas, se selecciona componentes prefabricados, se reestructura el modelo de objetos para atarlo a las metas de diseño y atributos de calidad y optimización del rendimiento.
- **Implementación. -** Los desarrolladores traducen el modelo de dominio de la solución a código ejecutable en el sistema, esto significa, implementar los atributos y métodos de cada objeto, e integrarlos de forma que funcionen como un solo sistema.

- **Pruebas –** Los desarrolladores buscan diferencias entre el sistema y los modelos ejecutando el sistema con datos de ejemplo, existen varios tipos de pruebas que se realizan, como son las pruebas de unidad, pruebas de integración, y pruebas del sistema, lo cual les permite asegurar que el sistema cumple con las especificaciones dadas.

1.4. Desarrollo profesional de *software*

La programación, hoy en día es una actividad que muchos profesionales realizan para resolver sus propias necesidades, sin embargo, los ingenieros en sistemas, Tecnologías de la Información, informática o carreras afines, hacen desarrollo de *software* de manera profesional, eso significa que no desarrollan para sí mismos, sino que desarrollan para solucionar las necesidades sobre todo de empresas de diferentes sectores.

En este sentido, debe tomarse prácticas y estrategias desarrollo a las que llamamos desarrollo profesional de *software*, además, evaluar el impacto de los fallos en los proyectos de ingeniería de *software* y las implicaciones éticas conjugan una serie de elementos fundamentales para comprender el papel del ingeniero en tecnologías de la información en lo referente al desarrollo de aplicaciones de *software*.

Estudie, ahora, toda la unidad 1 de la guía didáctica, la cual comprende los temas de Desarrollo profesional de *software*, Fallas de ingeniería de *software* y Ética en la ingeniería de *software*. Con esto se completa la unidad.

Actividad de aprendizaje recomendada

A continuación, le invito a que desarrolle las actividades de aprendizaje recomendadas:

1. Revise el siguiente video introductorio a la ingeniería de *software* para profundizar en sus conceptos básicos: Ingeniería de software clase 1.
2. Con base en lo explicado en el apartado 1.2.4 y el ejercicio de la tabla 2, piense en 4 requerimientos funcionales y 4 requerimientos no funcionales para un sistema de venta de boletos de autobús para transporte interprovincial. Anótelos en la siguiente tabla:
   Lista de requerimientos funcionales y no funcionales

ID	Requerimiento
REQUERIMIENTOS FUNCIONALES	
F01	
F02	
F03	
F04	
REQUERIMIENTOS NO FUNCIONALES	
N01	
N02	
N03	
N04	

Nota: por favor, complete la actividad en un cuaderno o documento Word.

3. Desarrolle la autoevaluación de la unidad 1 de la guía didáctica, recuerde que en ella también hay el solucionario para verificar sus respuestas.
4. Con base en la información provista en la presente unidad, desarrolle un ensayo de máximo 2 páginas, donde sustente el papel del ingeniero de *software* y la importancia de su trabajo para las empresas y para la sociedad en general.
5. En la presente unidad, se ha realizado la introducción a los conceptos más importantes de la ingeniería de *software*, desde su origen hasta establecer las actividades del proceso desde el punto de vista orientado a objetos. Con el fin de enfocarse en los aspectos más relevantes, le invito a que desarrolle la siguiente autoevaluación.

Autoevaluación 1

En las siguientes preguntas escoja la opción correcta:

1. El origen de la ingeniería de software como una ciencia surgió a partir de una reunión de la OTAN en la cual acuñaron el término como respuesta a la denominada crisis del software.
    a. El uso de computadoras en el sector empresarial.
    b. El surgimiento de la profesión de los programadores.
    c. El desarrollo de los primeros sistemas de tiempo compartido.
    d. La fabricación de ordenadores más pequeños.
2. En relación con la evolución histórica del desarrollo de software del OCW de Ingeniería de software, ¿en qué año se empezaron a utilizar los sistemas operativos y a qué era corresponden?
    a. 1960, 1.ª Era.
    b. 1970, 2.ª Era.
    c. 1990, 3.ª Era.
    d. 2010, Boom Tic.

3. En relación con las tendencias de mercado, ¿cuáles de las siguientes características corresponden a SOA?
    a. Smartphones potentes.
    b. Sistemas abiertos a través de servicios expuestos al exterior.
    c. Computación en la nube.
    d. Orientación a la gestión basada en procesos y proyectos.
4. ¿Cuál de las siguientes alternativas incluye todo lo que el término software representa?
    a. Programas, datos, documentación.
    b. Programas y datos.
    c. Programas y documentación.
    d. Programas, personas, procesos y documentos.
5. ¿Cuál de las siguientes características no es aplicable al software?
    a. Es un elemento lógico, no físico.
    b. No se puede estropear, pero se degrada.
    c. Se construye a la medida.
    d. No tiene incertidumbre en su desarrollo.
6. ¿Cuál de las siguientes reglas no se debe aplicar para un proyecto de software?
    a. No inventar rueda, use estándares.
    b. Utilice la ley del mínimo esfuerzo.
    c. Debe aprender de la experiencia propia o de otros a través de lecciones aprendidas.
    d. Desarrollar es un proceso secuencial, donde el cliente establece paso a paso lo que necesita.
7. La aplicación de la ingeniería de software, ¿cuál de los siguientes resultados se espera que produzca?
    a. Proyectos eficientes.
    b. Personal motivado.
    c. Software de calidad.
    d. Documentación amplia y efectiva.
8. Una de las estrategias utilizadas en Ingeniería de Software es el modelado, ¿con qué propósito se usa esta técnica?
    a. Reducir la complejidad.

b. Documentar visualmente el proyecto.
c. Compartir el conocimiento del proyecto.
d. Resolver problemas.

9. ¿En cuál de las siguientes circunstancias es recomendable utilizar modelos?
    a. Cuando los sistemas son pequeños y simples.
    b. Cuando los sistemas no tienen complejidad.
    c. Siempre debe usarse modelos.
    d. Cuando los sistemas son muy grandes y complejos.
10. ¿En qué etapa del proceso de desarrollo, los ingenieros de software construyen el modelo de dominio de aplicación?
    a. Durante la etapa de implementación.
    b. Durante la etapa de diseño del sistema.
    c. Durante el diseño de objetos.
    d. Durante las etapas de levantamiento de requerimientos y análisis.

Ir al solucionario

Resultado de aprendizaje 2:

Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.

Para alcanzar el resultado de aprendizaje 2, los estudiantes se familiarizan con las fases del ciclo de vida del desarrollo de sistemas. Examinarán los diversos modelos de procesos de *software* y las metodologías de desarrollo, tanto tradicionales como ágiles. Este conocimiento les permitirá identificar y aplicar adecuadamente cada fase del ciclo de vida en la gestión de proyectos de *software*, asegurando una planificación efectiva y una entrega exitosa del producto final.

Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.

Semana 2

Durante la primera semana, se realizó una introducción a lo que es la ingeniería de *software*, se analizaron algunos de los principios y se describió de manera breve algunas de las actividades de desarrollo desde la perspectiva Orientada a Objetos, sin embargo, en la práctica, los proyectos de desarrollo de *software* se abordan siguiendo un proceso de desarrollo de *software*, el cual consiste en primer lugar en un enfoque que en general puede ser considerado tradicional o predictivo y ágil.

En esta semana vamos a abordar los procesos de desarrollo de *software* más comunes, e intentaremos establecer los criterios necesarios para seleccionar uno u otro en función de las características del proyecto.

Como recursos de aprendizaje utilizaremos:

- Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL. La unidad 2. Procesos de desarrollo. En ella encontrará orientaciones de estudio y le llevará a usar otros recursos.
- El **texto básico** Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación.

Unidad 2. Procesos de software

Para el estudio del contenido vamos a utilizar como recurso principal la guía didáctica de Fundamentos de ingeniería de *software* y el **texto básico**.

2.1. Modelos de proceso de software

Un proceso de *software* es "un conjunto coherente de políticas, estructuras organizativas, tecnologías, procedimientos y artefactos que se necesitan para concebir, desarrollar, implantar y mantener un producto de *software*" y, un modelo de proceso es "una definición de alto nivel de las fases por las que transcurren los proyectos de desarrollo de *software*" (Sánchez et al., 2012), a los modelos de proceso también se los denomina como Ciclo de Vida del Desarrollo de *Software* (CVDS).

Para abordar este tema, sírvase revisar el apartado 2.1 de la guía didáctica con las correspondientes referencias a los materiales allí indicados, en ella se analizan los modelos de proceso cascada, el modelo incremental y el modelo de ingeniería de *software* orientado a la reutilización.

Para dar más luces sobre la selección de un modelo de proceso adecuado, es muy importante tener en cuenta el nivel de incertidumbre del proyecto, y para ello tomaremos como referencia la Guía de Agilidad (*Project Management Institute*, 2017) que establece 4 tipos de ciclos de vida y además recomienda un enfoque de acuerdo al nivel de incertidumbre de los proyectos como se indica en el siguiente módulo didáctico:

Ciclos de vida

Una vez completada la lectura de la guía y sus referencias se puede identificar una gran variedad de enfoques para abordar el desarrollo de una aplicación de *software*, no obstante, los desarrolladores pueden mostrarse indecisos al momento de seleccionar uno u otro enfoque, las tablas 1 y 2 de la guía didáctica, permiten evaluar algunos criterios antes de seleccionar un proceso de desarrollo.

En la tabla 4 de la presente guía, se presentan algunas de las características de cada una de estas categorías de los ciclos de vida.

Tabla 4
Características de las diferentes categorías de ciclos de vida.

Características				
Enfoque	**Requerimientos**	**Actividades**	**Entregas**	**Metas**
Predictivo	Fijos	Se ejecutan una sola vez durante todo el proyecto.	Una sola	Gestionar costos
Iterativo	Dinámicos	Se repite hasta que los resultados sean correctos.	Una sola	Solución correcta.
Incremental	Dinámicos	Se ejecuta una vez por incremento.	Entregas pequeñas frecuentes	Velocidad
Ágil	Dinámicos	Se repite hasta que sea correcto.	Entregas pequeñas frecuentes.	Valor del cliente a través de entregas y retroalimentación.

Nota. Adaptado de Project Management Body Of Knowledge and Agile Practice Guide, por Project Management Institute, 2017, Pennsylvania

2.2. Metodologías de desarrollo tradicionales

Una vez estudiados los diferentes modelos de proceso, habrá notado que las actividades genéricas para el desarrollo de *software* son análisis, diseño, implementación, pruebas, y entrega, sin embargo, existen muchas metodologías, sobre todo las que tienen enfoque ágil cuyas actividades, aunque tienen similitudes, tienen diferencias significativas.

Una metodología es "una forma de hacer o adoptar un modelo para ejecutar una tarea o un conjunto de tareas, de modo que el objetivo de la tarea se logre como se esperaba" (Ahmed & Bhanu, 2016), los cuales pueden ser aplicados en función de las circunstancias del trabajo que se tiene que desarrollar. Considere como ejemplo la situación de construir un condominio de departamentos, en función de los requerimientos se tiene dos opciones, por un lado, se puede optar por utilizar estructuras prefabricadas y luego ensamblarlas para construir el edificio, o puede realizar una construcción con el método tradicional en el cual se construye todo con los materiales y herramientas básicas, en este caso la meta es la misma y los enfoques para conseguirlo son diferentes. La decisión sobre qué método seleccionar depende de los requerimientos del método, así como de las consideraciones tales como el plazo de entrega, los costos, la seguridad y la disponibilidad de los recursos y la tecnología. En la misma forma, una aplicación de *software* puede ser desarrollada utilizando metodologías conocidas o cualquier otra disponible para los desarrolladores.

En este apartado estudiaremos algunas metodologías de desarrollo de *software* que trabajan bajo el esquema predictivo o tradicional, recuerde que no se trata de profundizar en ellas, sino de conocer cuál es su enfoque, sus etapas, sus ventajas y desventajas, de modo que usted pueda decidir cuál sería la más conveniente para un proyecto específico.

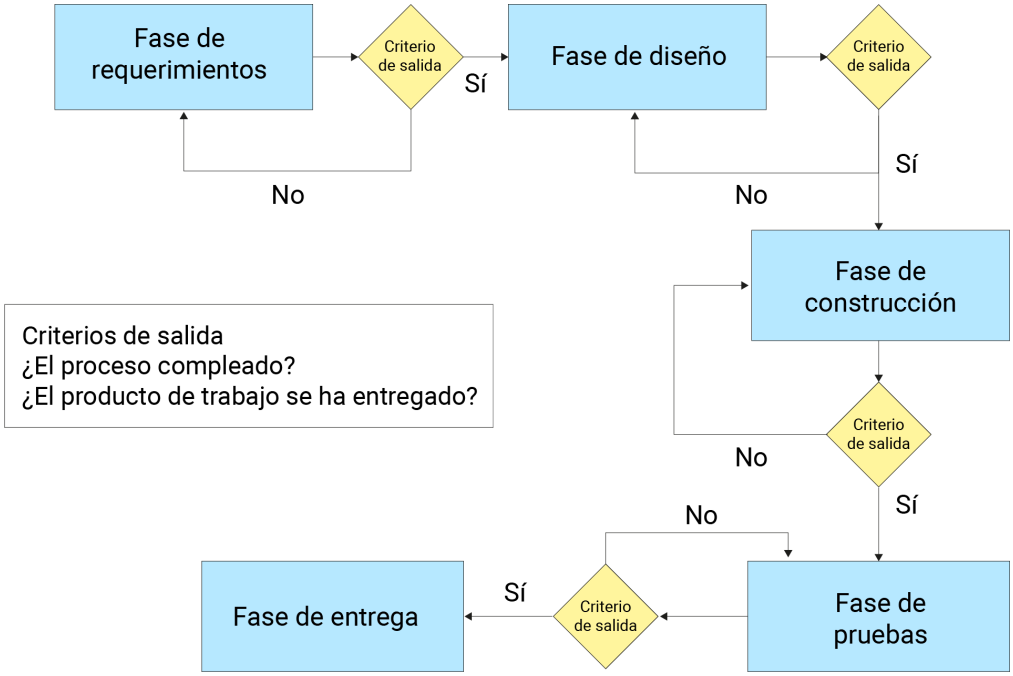
2.2.1 Cascada (waterfall model)

Fue documentado por primera vez por Benington Herber D., en 1956 y en 1970 fue modificado por Winston Royce (Ruparelia, 2010). El propósito de esta metodología era reducir los costos de desarrollo, la especificación inicia a muy alto nivel hasta llegar a los niveles más bajos, a este enfoque se lo conoce como *TOP*-d*OWN*. Esta metodología fue la primera un usarse formalmente para el desarrollo de *software*, promueve la operación secuencial de sus procesos con planes claramente definidos. Ello ha llevado a que se denomine como dirigida por un plan.

En el modelo original de la metodología, cada paso se realizaba de manera secuencial, y no se avanza al siguiente, a no ser que se cumplan los criterios de salida del anterior, esto se puede apreciar en la figura 1. Los criterios de salida están asociados a dos aspectos importantes, el primero es que los artefactos se hayan completado y el segundo es que estos hayan sido aceptados por parte del cliente.

Figura 1

Control de calidad en el modelo en cascada

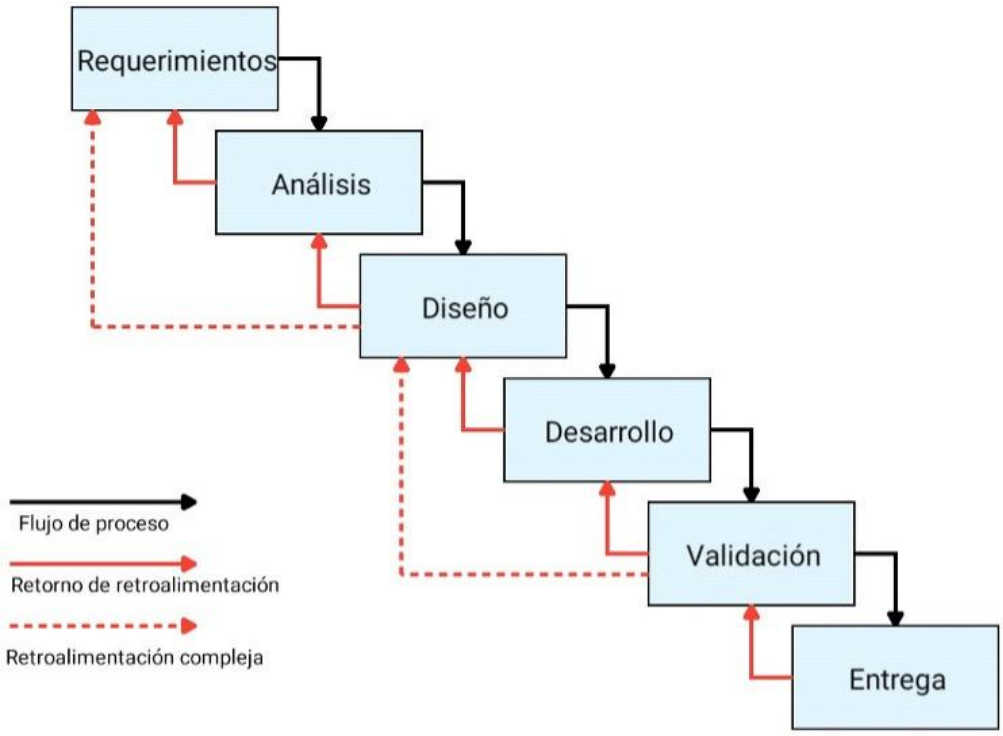


Nota. Adaptado de Foundations of Software Engineering. In National Conference Publication - Institution of Engineers [Ilustración], por Ahmed, A., & Bhanu, P., 2016. , Australia. Boca Raton: Taylor & Francis.

En la figura 2 se aprecia la versión de 1970 del modelo en cascada. La versión de Royce mejora el modelo original incluyendo bucles de retroalimentación de forma que la fase anterior puede ser revisitada.

Figura 2

Modelo en cascada de Royce



Nota. Adaptado de Software development lifecycle models [Ilustración], por Ruparelia, N., 2010, Digital Library, CC BY. 4.0.

El modelo en cascada opera de la siguiente manera:

- En la fase de requerimientos, el equipo del proyecto se enfoca en realizar un gran esfuerzo en recolectar todos los requerimientos del cliente y de los usuarios, luego son documentados en un proceso exhaustivo, finalmente son validados y aprobados por el cliente, con ellos se elabora el documento de Requerimientos de S*oftware*, y una vez completado se usa como insumo para la etapa de diseño.
- En la fase de **análisis**, se desarrollan especificaciones detalladas del sistema y se produce la documentación necesaria para que los diseñadores puedan realizar el diseño preliminar del sistema, en caso

de inconsistencias o de que algo no quedó claro, se puede regresar a la fase de requerimientos y modificar la documentación correspondiente, con ello se revisaría nuevamente el análisis. Este documento de análisis también tiene que ser aprobado antes de pasar a la fase de diseño.

- En la fase de **diseño**, los diseñadores elaboran varios modelos del sistema y generan la documentación correspondiente que consiste en: el documento de diseño de interfaces, diseño de componentes, diseño de datos. Al igual en la fase anterior, en caso de encontrar fallos en el diseño durante esta fase, se regresa a la fase de análisis y en un escenario más complejo, se podría regresar a la fase de requerimientos.
- En la fase de **desarrollo** se construye el código de la aplicación, el resultado es el *software* terminado, probado y documentado, la siguiente fase que es la validación.
- En la fase de **validación,** se realizan las pruebas correspondientes para determinar si el sistema cumple con las especificaciones, y en caso de que no lo haga, se puede regresar a la fase de desarrollo y si el problema es más complejo, se regresaría a la fase de diseño.
- Una vez que se han completado exitosamente estas etapas, el sistema se entrega al cliente y usuarios, se entregan manuales, se preparan los datos para la operación del sistema y se capacita a los usuarios para su uso.

Es importante mencionar que, si bien es un modelo sólido, no es la mejor alternativa, o al menos no para cualquier proyecto, a continuación, se resumen algunas de sus ventajas y desventajas:

Ventajas:

- Se planifica todo el proyecto al inicio, se establecen líneas base, hitos y plazos de entrega, ello facilita el monitoreo de proyecto, mejora la visibilidad de los procesos de desarrollo.

- Se puede obtener reportes de estado en cualquier momento durante la ejecución, y los interesados pueden tomar decisiones respecto de cualquier cambio para asegurar que el proyecto avanza sin contratiempos. Los reportes de estado pueden ser internos y externos, los externos son para los interesados, y con base en ellos pueden solicitar cambios.
- Soporta controles de calidad: al finalizar cada fase, los artefactos desarrollados se pueden probar, y solo pasa a la siguiente fase si los artefactos cumplen con las especificaciones de las pruebas.
- Es adecuado para el desarrollo de productos muy grandes: Permite que el equipo sea de cualquier tamaño, desde unas pocas personas hasta cientos de ellas, y de ser necesario un proyecto muy grande puede ser desarrollado en corto tiempo por muchas personas.
- El modelo implica mucha documentación, por tanto, si un miembro del equipo abandona el proyecto, puede usar la documentación para ponerse al tanto de lo realizado y continuar con lo nuevo.

Desventajas:

- Tiene una gran carga de trabajo en planificación, debido a que el producto se desarrolla por completo y no de manera incremental, y, por tanto, requiere muchas personas trabajando en el equipo.
- La carga operativa en cuanto a la gestión de personal es muy grande.
- La mayoría de las veces los usuarios no tienen conocimientos técnicos, por tanto, los requerimientos pueden tener muchas ambigüedades, y, en consecuencia, la planificación puede ser fallida.
- Todos los requerimientos deben congelarse al inicio del proyecto, ya que se usan como base para la planificación, los cambios en etapas avanzadas del proyecto resultan demasiado costosos, puesto que se necesita volver a etapas iniciales para implementarlos. En la realidad los clientes y usuarios suelen cambiar muchos requerimientos una vez iniciado el proyecto, por tanto, hoy en día es uno de los procesos menos convenientes.

Cuando usar la metodología:

- En proyectos en los cuales se requiere que el proyecto se termine de acuerdo a un cronograma preestablecido.
- Cuando se requiere documentación muy formal.
- Proyectos sencillos, con poca incertidumbre baja, y bajo nivel de riesgo.

Continuemos aprendiendo otras metodologías de desarrollo tradicionales.

2.2.2 Proceso unificado de desarrollo

El Proceso Unificado de Desarrollo (RUP por sus siglas en inglés), fue creado por Grady Booch, Ivar Jacobson, y James Rumabugh, en 1998 con el acrónimo de RUP (*Rational Unified Process*), cuando trabajan para Rational *Software* como respuesta a los problemas existentes en la época con los procesos de desarrollo existentes.

De acuerdo a (Kruchten, 2000) para crear RUP, sus autores se plantearon los principales problemas con el desarrollo de *software*, establecieron sus causas raíces y propusieron las mejores prácticas Rational de la ingeniería de *software*.

Los principales problemas o síntomas son los siguientes:

- Escasa comprensión de las necesidades de los usuarios finales.
- Imposibilidad de resolver los requerimientos cambiantes.
- Módulos que no empatan entre sí.
- Elsoftware muy difícil de mantener o extender.
- Descubrimiento tardío de fallas serias en el proyecto.
- Calidad del software muy pobre.
- Rendimiento del software
- Miembros del equipo que no colaboran entre sí, haciendo imposible saber reconstruir, o saber quién hizo, quién cambió, cuándo, cómo y por qué.

- Proceso de compilación, construcción y liberación no confiable.

Luego identificaron las causas raíces de esos problemas, que se pueden resumir en las siguientes:

- Gestión de requerimientos improvisada.
- Comunicación imprecisa o ambigua.
- Arquitectura débil.
- Complejidad excesiva.
- Inconsistencias no detectadas en los requisitos, diseños e implementaciones.
- Pruebas insuficientes.
- Evaluación subjetiva del estado del proyecto.
- Fallos al momento de enfrentar riesgos.
- Propagación de cambios descontrolada.
- Automatización insuficiente.

Para resolver estos problemas, los autores propusieron algunas prácticas de la ingeniería de software que se detallan en el siguiente módulo didáctico:

Prácticas de la ingeniería de software

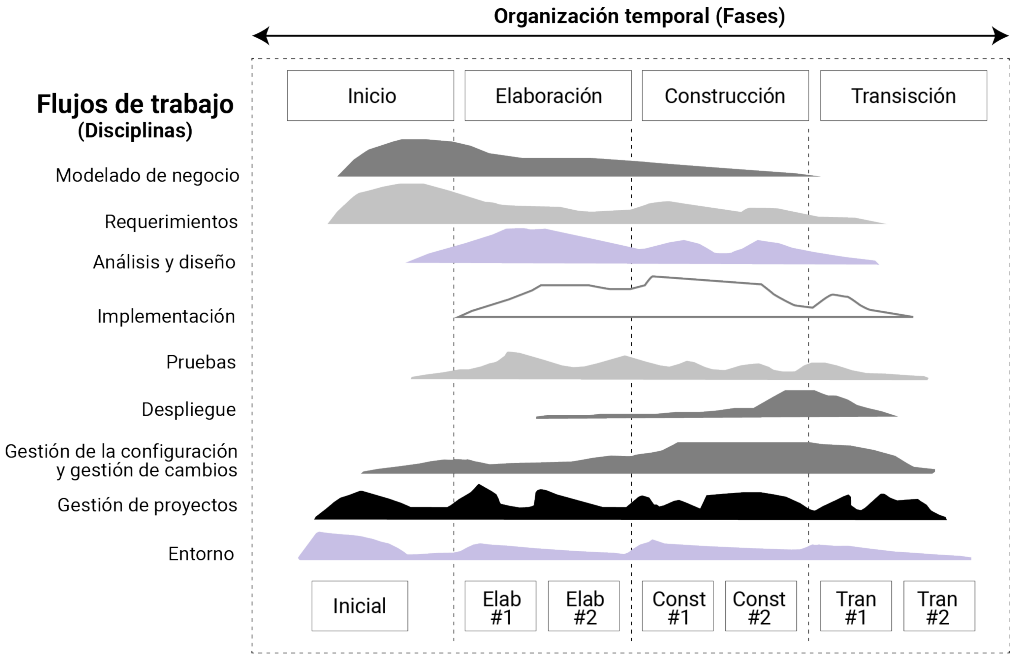
El RUP está organizado en dos dimensiones, horizontalmente podemos distinguir 4 fases: inicio, elaboración, construcción y transición, y verticalmente consta de nueve flujos de trabajo o disciplinas, en cada fase a partir de la segunda (elaboración) se desarrollan varias iteraciones, en las cuales se realizan todos los procesos establecidos en las disciplinas y producen en resultado ejecutable.

En la fase de inicio se busca definir el alcance del proyecto, y como artefacto resultante se obtiene un documento de visión, en la fase de elaboración se inicia el desarrollo, los artefactos resultantes son los programas, la documentación y el artefacto que marca el fin de la fase es la estabilización de la arquitectura de la aplicación, la tercera fase es la construcción, el resultado para el usuario son los componentes de la aplicación ya completos, la documentación asociada y en la fase de transición se

completan todos los entregables, la documentación, se capacita a los usuarios y se cierra el proyecto. Todas las disciplinas se ejecutan en todas las fases del proceso RUP, y la intensidad de trabajo requerido en cada disciplina, se ilustra con las montañas que recorren cada fase. Este proceso se ilustra en la figura 3.

Figura 3

El Proceso Unificado Rational (RUP)



Nota. Adaptado de The Rational Unified Process: An introduction [Ilustración], por Kruchten, P., 2000, Massachusetts: Addison Wesey

Ventajas:

- RUP admite el concepto de flujo de trabajo, y estos se pueden ejecutar en todas las iteraciones del proceso, por lo tanto, se produce un proceso iterativo incremental, que con cada iteración completa y mejora la aplicación resultante.

- La aplicación de las mejoras prácticas de la ingeniería de *software*, permite elevar la calidad de los resultados.

Desventajas:

- La ejecución de todas las disciplinas resulta muy pesada en términos de mantenimiento de las fases del proyecto, para trabajar en RUP se requiere un equipo grande de personas.
- RUP exige mucha documentación.

Cuando utilizar RUP:

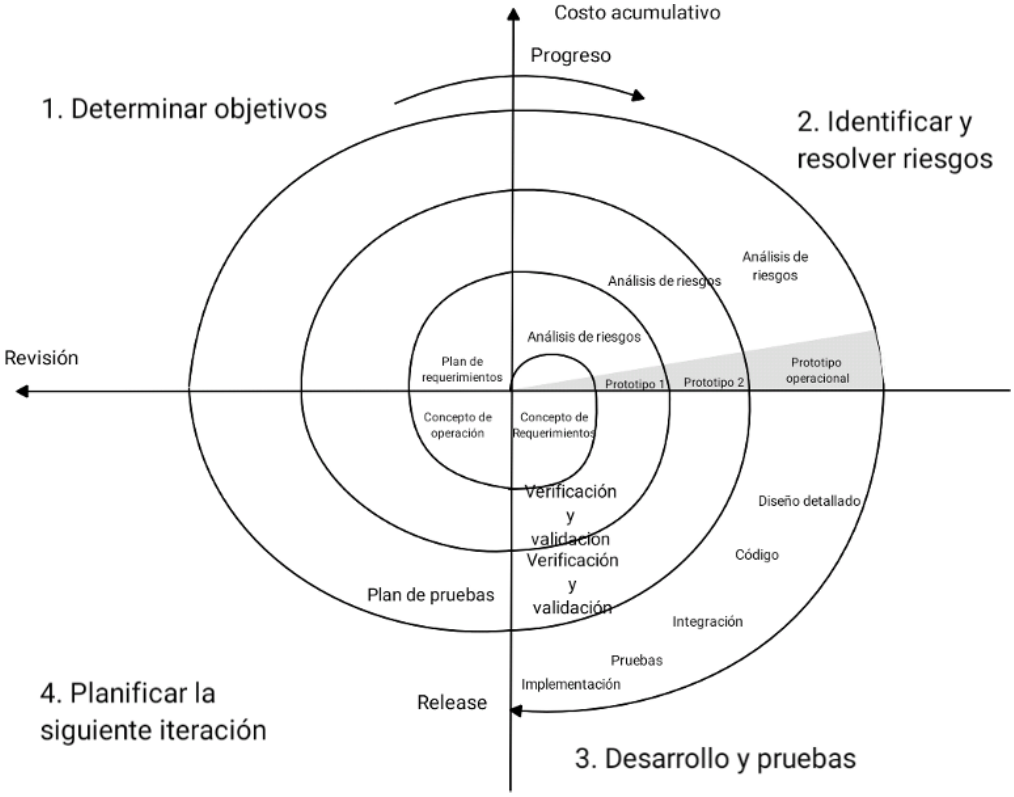
- El RUP es un proceso de desarrollo robusto, que al basarse en iteraciones permite refinar el producto conforme avanza en el tiempo.
- Es muy recomendable cuando los niveles de incertidumbre y riesgo son altos y se requiere mucha formalidad en el proyecto, esto debido a la documentación.
- A pesar de que incluye procesos iterativos e incrementales, RUP no se considera un proceso de desarrollo ágil, debido a la cantidad de documentación que exige y a la cantidad de roles que deben operar, sin embargo, esto lo hace ideal para proyectos muy grandes.

2.2.3 Modelo en espiral

En 1986, Boehm, modificó el modelo en cascada, introduciendo varias iteraciones que se surgen en espiral hacia afuera, comenzando con pequeños incrementos, aplicando la filosofía start, *small*, *think big* (Ruparelia, 2010). El modelo en espiral representa un cambio en el paradigma orientado a especificaciones del modelo en cascada a uno dirigido por riesgos.

Figura 4

El modelo en espiral de Boehm



Nota. Adaptado de Software development lifecycle models [Ilustración], por Ruparelia, N. B.., 2010, Digital Library, CC BY. 4.0

En la figura 4 se aprecia el modelo en espiral, en él cada ciclo pasa por cuatro cuadrantes que son:

1. Determinar objetivos.
2. Evaluar alternativas e identificar y evaluar riesgos.
3. Desarrollo y pruebas, y.
4. Planificar la siguiente iteración.

Conforme avanza cada ciclo, se construye un prototipo, que se verifica contra los requerimientos y es validado mediante pruebas. Previo a ello, se identifican y analizan los riesgos con el fin de gestionarlos, luego estos son categorizados como riesgos relacionados con el rendimiento o desarrollo. Si predominan los riesgos relacionados al desarrollo, el siguiente paso se realiza con el enfoque incremental del enfoque en cascada. Caso contrario, si predominan los riesgos de rendimiento, el siguiente paso es continuar con la espiral hasta el siguiente ciclo evolucionario. Esto asegura que el prototipo del segundo cuadrante tiene riesgos mínimos asociados a él.

En este modelo, la gestión del riesgo se usa como base para determinar el tiempo y el esfuerzo que se usará para todas las actividades durante el ciclo, y también se usa para controlar los costos.

De acuerdo a (Ahmed & Bhanu, 2016), el modelo en espiral no es un modelo ágil, ya que el modelo en espiral no produce una versión ejecutable del sistema en cada iteración, si no prototipos que se usan para demostrar al cliente lo que el equipo está desarrollando.

Ventajas:

- En el modelo en espiral, se considera como buena práctica el construir un buen prototipo antes de construir el producto final, mismo que es refinado durante cada iteración.
- El cliente puede dar su retroalimentación respecto de cada prototipo y provee más información sobre partes equivocadas o faltantes en el producto.
- Una vez que el cliente ha dado su aprobación sobre los prototipos, el equipo comienza a desarrollar el producto.

Desventajas:

- El modelo en espiral puede resultar costoso por las muchas iteraciones para obtener el producto de *software*.

- Puede haber productos que se desarrollen en 1 sola iteración con los cual este modelo sería excesivo.

Cuando utilizar el modelo en espiral:

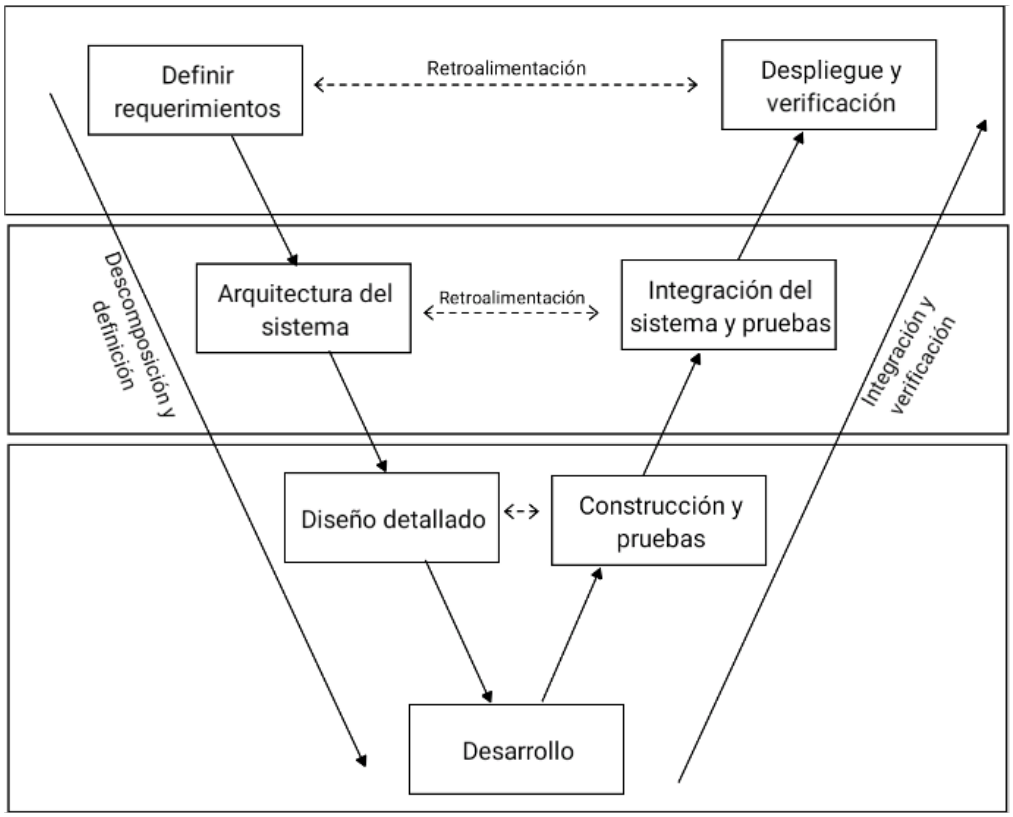
El modelo en espiral es conveniente utilizarlo para modelos en los que los niveles de riesgo son demasiado altos.

2.2.4 El modelo en V

Fue desarrollado por la NASA, se presenta como una variación del modelo en cascada se presenta en forma de una V, la parte de izquierda representa la evolución de los Requerimientos de Usuario en componentes más pequeños a lo largo del proceso de descomposición y definición, la parte derecha representa la integración de los componentes del sistema en niveles sucesivos de implementación y ensamblaje. En la figura 5 se aprecia este modelo. Visto verticalmente el modelo representa el nivel de descomposición del sistema, en la parte superior el sistema y en el fondo en nivel más bajo de descomposición.

Figura 5

El modelo en V



Nota. Adaptado de Software development lifecycle models [Ilustración], por Ruparelia, N., 2010, Digital Library, CC BY. 4.0.

Una característica del modelo en V es que es simétrico a lo largo de ambas ramas, por tanto, los procesos de verificación y aseguramiento de la calidad están definidos para la rama derecha durante el paso correspondiente en la rama izquierda, con ello es posible asegurar que los requerimientos y el diseño son verificables de forma específica, medible, alcanzable, realista y limitada en el tiempo (SMART por sus siglas en inglés).

Ventajas:

- Introduce verificaciones de cada una de las fases, lo cual facilita la localización de fallos, esto lo hace mucho más robusto y completo que el modelo en cascada.
- El modelo es sencillo y fácil de aprender.
- Se hace explícita la iteración y el trabajo que se debe realizar, lo que produce un software de mejor calidad que en cascada.
- El cliente puede dar su retroalimentación y se involucra en las pruebas.

Desventajas:

- A pesar de la retroalimentación del cliente, es difícil que el cliente exponga todos los requisitos.
- Al igual que en el modelo en cascada, el producto se entrega al finalizar el ciclo de vida.
- La metodología no contempla la posibilidad de regresar a etapas inmediatamente anteriores, lo cual sí puede ser necesario.

Cuando utilizar el modelo en V:

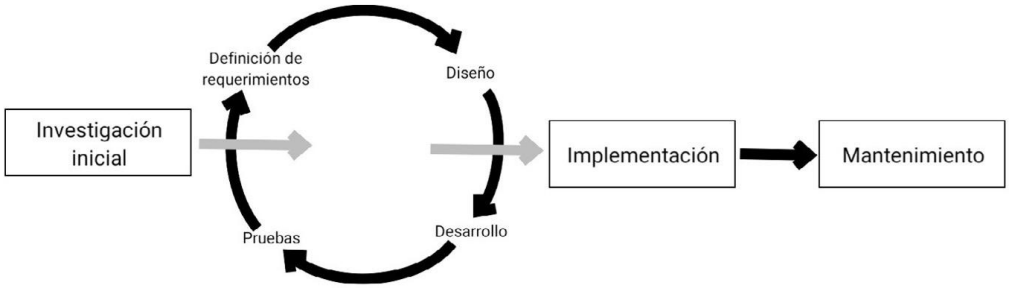
El modelo en V es una mejora al modelo en cascada, por lo que su uso es recomendado para aplicaciones de complejidad en incertidumbre baja, pero que requieran alta confiabilidad.

2.2.5 Desarrollo Rápido de Aplicaciones (RAD)

Desarrollada por James Martin en 1991, utiliza prototipos como mecanismo de desarrollo iterativo, promueve una atmósfera colaborativa donde participan activamente los interesados del negocio en la creación de prototipos, casos de prueba y ejecutando pruebas unitarias. El esquema de trabajo de RAD se ilustra en la figura 6.

Figura 6

Metodología Rapid Application Development (RAD)



Nota. Adaptado de Software development lifecycle models [Ilustración], por Ruparelia, N., 2010, Digital Library, CC BY. 4.0

Uno de los aportes más significativos ha sido la inclusión de procesos iterativos e incrementales, gracias a ellos se consigue entregar valor al cliente lo más pronto posible y el equipo consigue entregar a tiempo un *software* que se adapta a las necesidades de los usuarios, a diferencia de los modelos predictivos o tradicionales que retrasan las entregas a los usuarios hasta el final del proyecto, con lo cual muchas cosas pudieron cambiar en el entorno del cliente.

Quedan sin mencionar metodologías como el modelo basado en prototipos, Análisis y Diseño Orientado a Objetos (OOAD) de Grady Booch y Object, *Object Modeling Technique* (OMT), las cuales han servido de base para el desarrollo del Lenguaje Unificado de Modelado y el Proceso Unificado de Desarrollo.

Resultado de aprendizaje 2:

Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 3

Unidad 2. Procesos de software

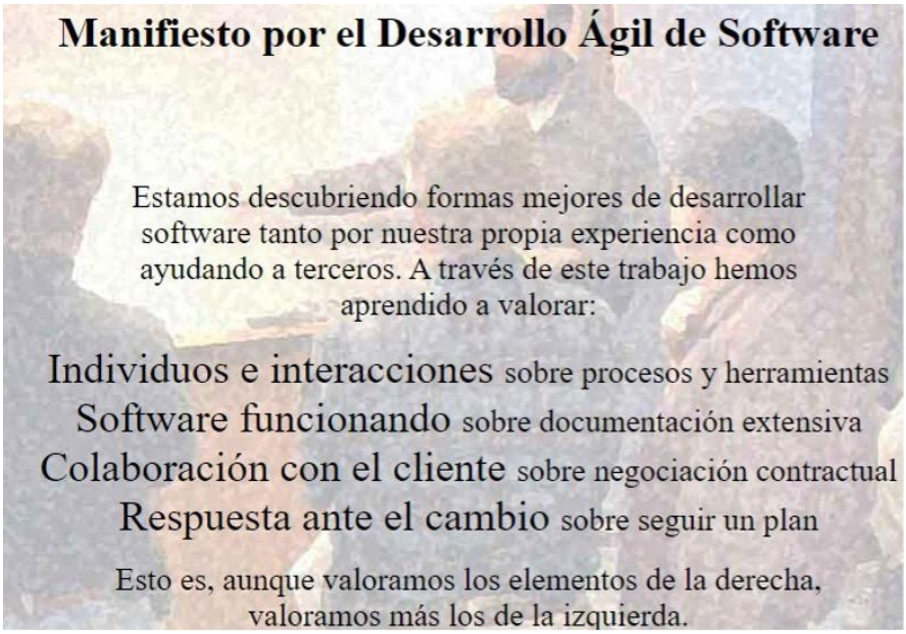
2.3. Metodologías ágiles

En la década de los 90, el uso de procesos de desarrollo predictivos se había extendido y funcionaban bien con proyectos muy grandes y costosos, sin embargo, había mucho descontento por los tiempos de entrega y la cantidad de cambios que surgían en los clientes y usuarios. Para (Ahmed, 2018), la presencia de *Internet* hizo que muchas empresas consideran su potencial para los negocios, y contrataron el desarrollo de muchos sitios *web*, que con los métodos tradicionales no facilitaban el trabajo, esto llevó a la comunidad de desarrolladores a pensar en métodos alternativos de desarrollar el trabajo que reduzcan los tiempos de entrega y eliminen obstáculos presentes en las metodologías tradicionales.

Esta situación llevó a un grupo de desarrolladores a crear una filosofía denominada Manifiesto Ágil, el cual establece los lineamientos y los principios sobre cómo debería ser el desarrollo de *software*. En la figura 7 se muestra una captura de pantalla del mismo, en él se expresan las aspiraciones de lo que los procesos de desarrollo ágiles deben tener para resolver los problemas de las metodologías tradicionales.

Figura 7

Manifiesto Ágil



Nota. Tomado de Manifiesto por el Desarrollo Ágil de Software [Fotografia], por Beedle, M., et al., 2001, agilemanifesto, CC BY. 4.0.

Si analizamos brevemente este manifiesto, vemos que se resalta y se prioriza a los clientes, entregando resultados útiles antes que documentación, además la colaboración con el cliente ayuda a que el proceso se adapte rápidamente a los cambios, de modo que el producto siempre agrega valor al cliente y usuarios.

Un peligro al tratar esta filosofía es el de pensar que se trata de procesos informales y sin calidad, sin embargo, es necesario acotar que no tienen nada de ello, simplemente se enfocan en los resultados antes que en la documentación o los procesos, pero la calidad del producto resultante debe ser la misma que con un enfoque tradicional, naturalmente el hecho de que se adapte rápidamente a los cambios hará que el resultado genere mayor

valor para los clientes, y más importante aún al colaborar con el cliente y los usuarios se logra que el producto resultante sea mucho más valioso para el cliente.

Ahora, le invito a revisar los principios del Manifiesto Ágil y conteste las siguientes preguntas, estos principios están en Principios del manifiesto ágil.

1. ¿Cuál es el tiempo preferido en el cual se debe entregar resultados?
2. ¿Cuál debería ser la mayor fortaleza de los equipos de trabajo?
3. ¿Cómo debe ser la comunicación entre miembros del equipo?
4. ¿Cómo abordan el tema de la calidad del producto?

Nota: por favor, conteste las preguntas en un cuaderno o documento Word.

Ahora que ha analizado los fundamentos de las metodologías ágiles, remítase a la guía didáctica, y revise el apartado 2.3.

A continuación, vamos a revisar brevemente dos de las metodologías ágiles más importantes.

2.3.1 Programación extrema (extreme programming)

Fue desarrollada por Kent Beck en 1996, la Programación Extrema (XP) es uno de los primeros modelos de desarrollo de *software* ágiles. Su nombre surge del hecho de que en esta metodología el desarrollo se lleva al extremo, es usual que los programadores primero prueben su lógica de negocio antes de escribir cualquier código, y para asegurar que la misma se implementa correctamente trabajan en pares, mientras una persona escribe el código, la otra lo revisa simultáneamente.

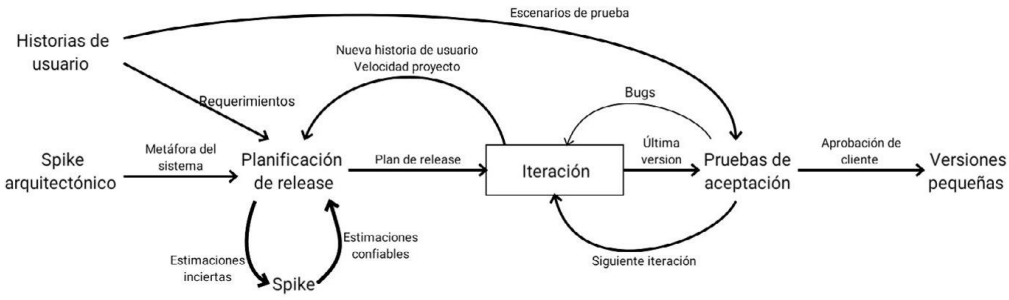
Para (Ahmed & Bhanu, 2016) la metodología Programación Extrema permite a las compañías liberar sus productos lo antes posible, y examinar el mercado con un producto con pocas características, en caso de que no sea aceptado, se puede detener su desarrollo.

Un factor clave en esta metodología es que el cliente siempre está involucrado y es responsable de todos los aspectos relacionados con la especificación de requerimientos.

Las actividades de XP son: codificación, en la cual se genera código; pruebas, se considera que es la mejor forma lograr un producto de calidad; escucha, en la cual los desarrolladores escuchan lo que el cliente espera del producto, explica la lógica de negocio que está detrás de las características que espera del producto, y, diseño, que cuando el producto ya es grande después de muchas iteraciones es una tarea primordial para garantizar su correcto funcionamiento. En la figura 8 se aprecia la forma de trabajo de XP.

Figura 8

Flujo de procesos de XP



Nota. Adaptado de Extreme Programming: A gentle introduction [Infografía], por Don Wells, 2013, extremeprogramming, CC BY 4.0

Ventajas

Permite co-localizar pequeños equipos de proyectos para trabajar en proyectos de desarrollo de *software*, no hay labores de gestión, debido a que los equipos se autoorganizan y solo hay un gestor de proyecto involucrado. Permite la construcción incremental de los productos de software, lo cual le permite incluir en el mercado productos de *software* muy temprano.

Desventajas

Si se requiere un producto muy grande construido de manera rápida, puede no ser posible con XP, considere que un equipo de 5 a 6 personas trabajando con XP pueden producir 20 líneas de código por día, eso significa que para un producto que requiera 100.000 líneas de código se requerirán aproximadamente 17 años asumiendo años de 300 días laborables(Ahmed & Bhanu, 2016).

Cuando usar XP

Se debería usar cuando se necesita desarrollar una aplicación de forma incremental, en esta metodología no es necesario esperar a tener todos los requerimientos y sobre todo cuando hay riesgo de que el cliente cambie los requerimientos de manera muy frecuente. Es decir, el nivel de incertidumbre es muy alto. Es más conveniente utilizarla cuando se tiene pequeños equipos que trabajan en el mismo sitio debido a que se necesitan muchas reuniones y actualizaciones del producto muy frecuentes.

2.3.2 Scrum

Varios autores consideran a SCRUM como la metodología ágil más popular para proyectos de *software*, sin embargo (Schwaber & Sutherland, 2017), definen a SCRUM como "Un *framework* en el cual las personas pueden abordar problemas complejos adaptativos, al tiempo que ofrecen productos

productivos y creativos del mayor valor posible". Su popularidad ha aumentado debido a su capacidad inherente para simplificar las cosas en un proyecto, así como para escalar cuando sea necesario.

SCRUM trabaja con 3 roles que son: SCRUM Master, el propietario del producto (*Product Owner*) y equipo del proyecto (*Team*).

El papel del SCRUM Master es facilitar el trabajo del equipo, resolviendo obstáculos que les impidan lograr su objetivo, su labor no es de gestor del proyecto, ya que, por principio de agilidad, los equipos son autoorganizados.

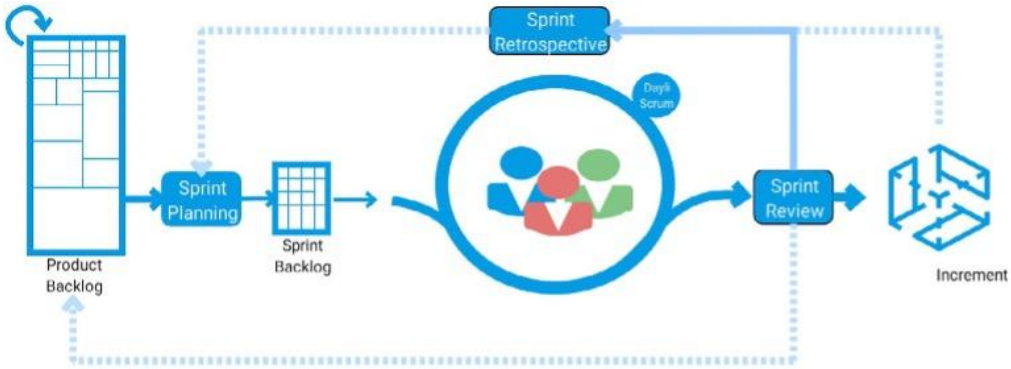
El *Product Owner* es el representante del cliente, su función es proporcionar los requisitos de *software* en forma de historias de usuario (*User Story*) al equipo del proyecto, también lleva a cabo pruebas de aceptación al final de cada iteración (*Sprint*) para asegurarse de que las características integradas del producto integradas en el *Sprint* se cumplen según las historias de los usuarios.

El equipo del proyecto es responsable de crear diseños de *software*, diseños de bases de datos, escribir código fuente y probar el código fuente. No hay especialistas en el equipo y cualquiera puede hacer el trabajo del proyecto. Sin embargo, es una buena idea tener una persona que sea propietaria de cada artefacto del proyecto. Entonces, aunque otros miembros del equipo también pueden crear artefactos del proyecto; la responsabilidad de cada artefacto del proyecto está claramente establecida.

El proyecto comienza con una épica (*Epic*) que es una gran historia creada por el cliente y que describe lo que se propone que haga el *software*, luego el proceso se desarrolla como se muestra en la figura 9, en la que se aprecia el *framework* de SCRUM.

Figura 9

Framework Scrum



Nota. Adaptado de Whay is scrum? [Infografía], por Scrum, 2020, SCRUM, CC BY 4.0.

A continuación, se explica brevemente el funcionamiento del *framework*:

1. Se crea una lista de historias de usuario que incluyen todo lo que se conoce que debe incluir el proyecto en un artefacto llamado pila de pendientes del producto (Product Backlog), esta lista evoluciona en la medida que avanza el proyecto y se aclaran los requerimientos.
2. Del *Product Backlog*, se toman grupos de historias de usuario y se desarrollan las iteraciones (*Sprints*), cada una dura alrededor de un mes, dando como resultado un incremento del producto (*Product Increment*)
3. Cada *Sprint* inicia con la planificación (*Sprint Planning*) en ella que establece su objetivo (*Sprint Goal*) y se determina lo que se puede entregar en ese incremento, produciendo otro artefacto que se conoce como pendientes del sprint (*Sprint Backlog*)
4. El equipo trabaja en el desarrollo del producto, al final de cada día se llevan a cabo reuniones diarias denominadas Scrum diarios (*Daily Scrum*), para evaluar el progreso hacia el objetivo del *Sprint* y monitorear el *Sprint Backlog*, ello permite optimizar el trabajo para alcanzar el *Sprint Goal*.

5. Al finalizar cada *Sprint*, se lleva a cabo una reunión denominada revisión del *sprint* (*Sprint Review*), en la que participan el SCRUM Master, el Team, el Product Owner y otros interesados clave, en ella el Product Owner explica los elementos de la lista que se han terminado y cuáles han quedado pendientes, el equipo realiza una demostración del incremento, y el *Product Owner* lo aprueba, si algo queda pendiente se agrega a la lista de pendientes, además se establecen las metas para las próximas iteraciones.
6. Una vez aprobado el incremento se lleva a cabo una reunión del equipo denominada retrospectiva del *sprint* (*Sprint Restrospective*) en la cual el equipo se inspecciona a sí mismo y crea un plan de mejoras para las próximas iteraciones, el SCRUM Máster se asegura de que el evento se lleve a cabo y de que los asistentes entiendan su propósito.

Ventajas de scrum

El desarrollo es iterativo e incremental, por tanto, al final de cada iteración se entrega un reléase del producto, lo cual significa que en corto tiempo se puede colocar en el mercado una versión menor del producto.

La planificación es un factor importante en Scrum, es muy fácil adaptar el proceso y entrar en mejora continua debido a las reuniones diarias, a la retrospectiva del *sprint*, y a la retroalimentación de parte cliente a través del *Product Owner*, lo cual permite que el producto esté siempre adecuado a las necesidades del negocio.

A través de una técnica conocida como Scrum of Scrums, un clúster de equipos de proyecto puede trabajar en el mismo proyecto, por tanto, se puede escalar proyectos muy grandes y complejos.

Desventajas

Scrum está pensado para trabajar con pequeños equipos de proyecto trabajando en el mismo espacio, por tanto, no es conveniente para equipos de trabajo muy grandes a no ser que constituyan en un esquema Scrum of Scrums.

Los proyectos que en los que se ha establecido costo y tiempo fijos, no pueden ser adaptados al entorno de Scrum, debido a que no es posible planificar todo el proyecto.

Para trabajar con Scrum es necesario que los miembros del equipo sean experimentados, de otra manera no podrían autoorganizarse, ya que no hay un gestor del proyecto, si no es así, el proyecto puede terminar en caos.

Si se requiere una persona con habilidades muy especializadas, puede resultar muy difícil trabajar con Scrum, puesto que se asume que todos los miembros del equipo son capaces de hacer cualquier clase de trabajo.

Cuando usar scrum

Cuando se debe abordar proyectos muy complejos con altos niveles de incertidumbre, se cuenta con equipos de trabajo altamente experimentados y además se cuenta con el apoyo y la retroalimentación del cliente a lo largo de todo el proceso.

Se debe considerar además proyectos en los que el plazo y el costo no se hayan establecido como fijos, ya que la dinámica de Scrum hace muy difícil planificar en esos términos. Como mucho, se puede saber qué es lo que se planifica en cada *Sprint*.

Estimado estudiante, si usted desea más información puede ingresar y registrarse de manera gratuita a los sitios de Agile Alliance y Scrum para descargar libros y material de capacitación de uso libre.

Actividades de aprendizaje recomendadas

Reforcemos el aprendizaje resolviendo las siguientes actividades:

1. Elabore una lista de criterios que, con base en las características del proyecto, le permitan optar por un enfoque predictivo, iterativo, incremental o ágil.
2. Elabore un cuadro comparativo entre las metodologías de desarrollo tradicionales donde se evidencie el ciclo de vida, ventajas y desventajas.
3. Compare las metodologías XP y SCRUM, determine los escenarios en los cuales resultaría más conveniente una u otra.
4. Desarrolle los ejercicios 2.1 a 2.10 del **texto básico**, considere la información adicional presentada en esta guía.

   Nota: por favor, complete las actividades en un cuaderno o documento Word.

5. Revise los siguientes vídeos:
   - #1. Qué son las metodologías tradicionales en el desarrollo de software.
   - #2. Qué son las metodologías ágiles en el desarrollo de software. Este video le ayudará para complementar su conocimiento sobre el origen y lo que son las metodologías ágiles.
   - #3. Scrum en 6 minutos - metodologías ágiles. Este vídeo le servirá para comprender los usos y el funcionamiento de Scrum.
6. Desarrolle la autoevaluación de la unidad 3 de la guía didáctica de Fundamentos de ingeniería de software.
7. Realice la siguiente autoevaluación para comprobar sus conocimientos.

Autoevaluación 2

En las siguientes preguntas escoja la opción correcta:

1. En el modelo en Cascada, ¿en qué etapa se establecen los servicios, las restricciones y las metas del sistema?
    a. Análisis y definición de requerimientos.
    b. Diseño del sistema y del software.
    c. Implementación y prueba de unidad.
2. ¿En qué modelo la siguiente fase no debe comenzar sino hasta que termine la fase previa?
    a. Cascada.
    b. Incremental.
    c. Orientado a la reutilización.
3. Diseñar una implementación inicial, exponer esta al comentario del usuario y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado, se conoce como:
    a. Cascada.
    b. Incremental.
    c. Orientado a la reutilización.
4. Las actividades principales en el proceso de ingeniería de requerimientos son:
    a. Especificación, análisis y diseño, implementación y pruebas.
    b. Análisis de componentes, modificación y diseño del sistema.
    c. Estudio de factibilidad, obtención y análisis, especificación y validación.
5. Uno de los componentes que se puede utilizar en un proceso orientado a la reutilización es:
    a. MVC.
    b. Servicios web.
    c. Sistemas remotos.

6. En la etapa de Especificación del software, la actividad que verifica que los requerimientos sean realistas, coherentes y completos, se conoce como:
    a. Pruebas unitarias.
    b. Validación del software.
    c. Validación de requerimientos.
7. ¿En qué actividad se establece la estructura global del sistema?
    a. Diseño arquitectónico.
    b. Diseño de interfaz.
    c. Diseño de componentes.
8. ¿En qué actividad se diseñan las estructuras del sistema de datos?
    a. Diseño de interfaz.
    b. Diseño de componentes.
    c. Diseño de la base de datos.
9. La etapa donde las personas que desarrollan el sistema ponen a prueba los componentes que construyen el sistema, se conoce como prueba de:
    a. Desarrollo.
    b. Sistema.
    c. Aceptación.
10. En el proceso de ingeniería de requerimientos, qué estrategia ayuda con la selección y validación de requerimientos del sistema:
    a. Obtención.
    b. Prototipo.
    c. Taller.

Ir al solucionario

Resultado de aprendizaje 3:

Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de software.

Para alcanzar el resultado de aprendizaje 3, se estudiará la definición de proyecto, su organización, las tareas y productos de trabajo asociados, y la creación de un cronograma. Con este conocimiento, los estudiantes podrán elegir y aplicar el ciclo de vida del desarrollo que mejor se ajuste a las necesidades del proyecto, garantizando una gestión eficaz y el logro de los objetivos establecidos.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 4

En la unidad 2 se estudiaron los modelos de proceso de *software* y algunas de las metodologías de desarrollo más importantes, vale aclarar que, si bien las metodologías ágiles están de moda, no todos los proyectos deben desarrollarse utilizando una de ellas, por ello es importante que el ingeniero en tecnologías de la información, sea capaz de discernir cuál enfoque es el más adecuado de acuerdo a las características del proyecto. En esta unidad se estudiará los conceptos y la estructura mínima de lo que es un proyecto, para ello vamos a usar como material bibliográfico los siguientes materiales:

Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL. La unidad 2. Procesos de desarrollo. En ella encontrará orientaciones de estudio y le llevará a usar otros recursos como el **texto básico**. Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación. Semana 4.

Unidad 3. Proyectos de software

Una de las características del desarrollo de *software* es que todo debe desarrollarse bajo la figura de un proyecto, esta concepción es importante porque rompe el esquema de trabajo de un programador.

En la presente unidad, vamos a resumir algunas de las características más relevantes de lo que es un proyecto de *software*, acotando que la gestión de proyectos como tal tiene un alcance mucho más grande del que trataremos aquí, de hecho, hay otra asignatura que se hará cargo de ese tema.

3.1. Definición de proyecto

Existen muchas definiciones de lo que es un proyecto, pero dado que el Cuerpo de Conocimiento de la Ingeniería de *Software* (Society, 2014), acoge el estándar del *Project Management Institute* (*Project Management Institute*, 2017), vamos a tomar la definición de ese estándar que establece en términos generales "Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único", este concepto aplica para cualquier tipo de producto, en este caso al *software*, y, por tanto, este esfuerzo tiene un objetivo, tiene unos recursos y tiene algunas restricciones de alcance, costo y tiempo. Ello nos lleva a la conclusión de que los trabajos que tienen que ver con el desarrollo, modificación o mantenimiento de un producto de *software*, se constituyen en un proyecto.

Un proyecto, por lo tanto, tiene un grupo de personas a los que se denomina el **equipo del proyecto** y un grupo de participantes que son los miembros del equipo. Las personas que se verán afectadas por el desarrollo del proyecto o tienen participación en el proyecto o en su resultado se denominan **interesados** (*stakeholders*), el equipo de proyecto debe contar con ellos para el desarrollo, ya que ellos aportan necesidades y pueden tomar decisiones respecto de la continuidad del proyecto.

Los participantes del equipo deben relacionarse de diferentes maneras y distinguimos algunas relaciones:

- **Reporte,** en las cuales los participantes informan el estado de avance de sus tareas.
- **Decisión,** en la que la persona a cargo o líder toma decisiones respecto del proyecto, y.
- **Comunicación,** que se utiliza para intercambiar información con otros participantes, usuarios, clientes. La comunicación puede ser jerárquica, en cuyo caso será unidireccional y horizontal cuando se da entre miembros del equipo.

El PMI (*Project Management Institute*, 2017) define dos conceptos importantes con relación al alcance, el **alcance del producto** que es el "conjunto de características y funciones de un producto, servicio o resultado" y el **alcance del proyecto** que es "el trabajo realizado para entregar un producto, servicio o resultado con las funciones y características especificadas", además se afirma que en ocasiones el alcance del proyecto incluye el alcance del producto.

3.2. Organización del proyecto

Un aspecto importante del proyecto es poder dar respuesta a preguntas como ¿quién es el responsable de qué parte del sistema?, ¿a quién se debe contactar cuando ocurren problemas con alguna versión específica de un componente?, ¿cómo se debe documentar un problema?, ¿cuáles son las restricciones del proyecto?, ¿cuáles son los criterios para evaluar el sistema?, ¿a quién se debe informar si aparecen nuevos requerimientos?, ¿quién es el encargado de hablar con el cliente? Según (Bruegge & Dutoit, 2010) desde la perspectiva de los desarrolladores un proyecto consta de 4 componentes y de 4 etapas, los componentes son:

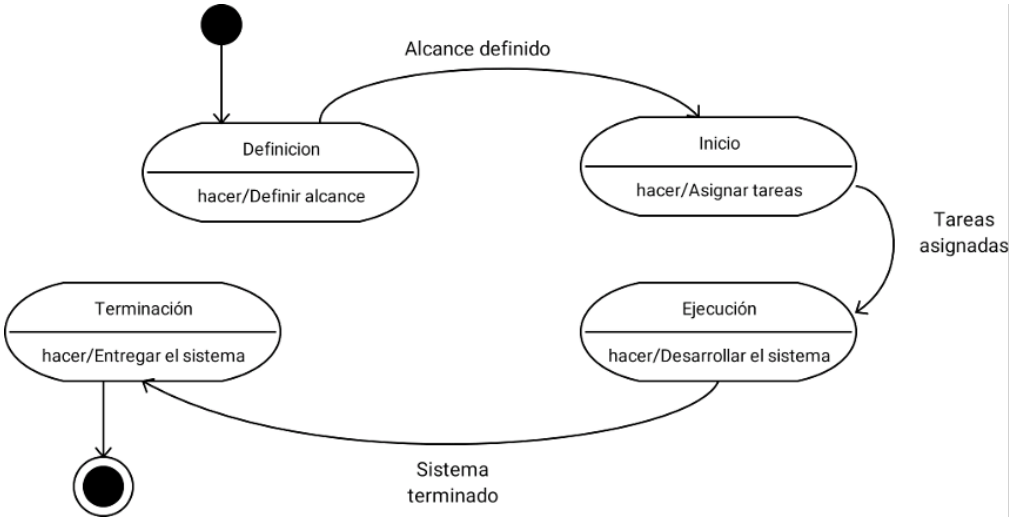
- **Producto de trabajo:** cualquier ítem producido por el proyecto, tal como una pieza de código, un modelo o un documento, los productos de trabajo que van a manos del cliente, se denominan

- **Cronograma:** artefacto que especifica cuándo se debe completar el trabajo, puede representarse en diagramas de Gantt.
- **Participante:** es cualquier persona que participa o cumple una función en el proyecto.
- **Tarea:** es el trabajo que debe ser ejecutado por un participante del proyecto para crear un producto de trabajo.

Un proyecto se organiza en etapas o fases, cada una de las cuales tiene un propósito bien definido, la culminación de una fase se produce cuando se completa una condición de salida. El modelo genérico de las fases de un proyecto se indica en la figura 10.

Figura 10

Diagrama de estados de las etapas de un proyecto de software



Nota. Adaptado de Object-Oriented Software Engineering [Ilustración], por Bruegge, B., & Dutoit, A. H., 2010. USA: Prentice Hall.

Veamos el funcionamiento de cada etapa de este modelo genérico de gestión de proyectos.

- **Etapa de definición:** el gestor del proyecto, el cliente, un miembro clave del proyecto, y el arquitecto de *software* trabajan en la comprensión

inicial de la arquitectura del sistema, específicamente los subsistemas que conformarán el sistema, el cronograma, el trabajo requerido y los recursos necesarios, esto se documenta en 3 artefactos: la declaración del problema, el documento de arquitectura inicial y el plan de gestión del proyecto.

- **Etapa de inicio:** el gestor del proyecto define una infraestructura, contrata a los participantes, organiza los equipos de trabajo, define los hitos principales y lanza el proyecto.
- **Etapa de ejecución:** los participantes desarrollan el sistema, reportan sus avances al líder de equipo, quien lleva a cabo el seguimiento del estado del trabajo de los desarrolladores e identifica posibles problemas, los líderes de equipo reportan el estado de avance al gestor del proyecto para evaluar el estado global del proyecto.
- **Etapa de terminación:** el resultado del proyecto se entrega al cliente y se guarda la historia del proyecto.

En un proyecto los participantes cumplen con un rol, el cual define un conjunto de tareas técnicas y de gestión que se espera de los participantes o del equipo, los roles pueden ser de gestión, de desarrollo y cruzados.

- **Roles de gestión:** son roles cuya función principal tiene que ver con la organización y ejecución del proyecto dentro de las restricciones, por ejemplo, se tiene el rol de gestor del proyecto y líder de equipo.
- **Roles de desarrollo:** tiene que ver con las actividades técnicas de desarrollo del sistema, estos incluyen: analistas, arquitectos de *software*, diseñadores y programadores.
- **Roles cruzados:** roles encargados de la coordinación entre equipos, son responsables de diseminar la información entre la estructura de comunicación del proyecto de un equipo a otro, y en ocasiones puede actuar como representante de un equipo a cargo de un subsistema, entre estos roles tenemos: Editor de documentación, gestor de la configuración y el *tester.*
- **Roles de consulta:** encargados de proveer soporte temporal en áreas donde los participantes del proyecto carecen de experiencia, la mayoría de las veces los usuarios y el cliente actúan como consultores

en el dominio de aplicación, también puede haber consultores técnicos expertos en determinadas tecnologías o métodos y consultores no técnicos que ayudan a direccionar aspectos legales y de mercadotecnia.

Podemos mencionar los siguientes roles: cliente, usuario final, especialista de dominio de aplicación, especialista de dominio de solución.

3.3. Tareas y productos de trabajo

Una tarea se entiende como una asignación de trabajo bien definida asignado a un rol. Los grupos de tareas relacionadas se denominan **actividades.** Estas tareas son asignadas por el gestor del proyecto o el líder de equipo a un participante, y además monitorea su avance hasta la culminación.

Un producto de trabajo es un ítem tangible resultante de una tarea, como por ejemplo un modelo de objetos, un diagrama de clases, un código fuente, un documento o partes de él. Estos productos de trabajo están sujetos a fechas de vencimiento y muchos se alimentan de otras tareas. Los productos de trabajo que no son visibles para el cliente, se denominan **productos de trabajo internos**, y los que se entregan al cliente se denominan **entregables.**

La especificación del trabajo que debe completarse, se define en un **paquete de trabajo**, el cual incluye el nombre de la tarea, la descripción, los recursos necesarios, las dependencias o entradas y las salidas, así como las dependencias con otras tareas.

3.4. Cronograma del proyecto

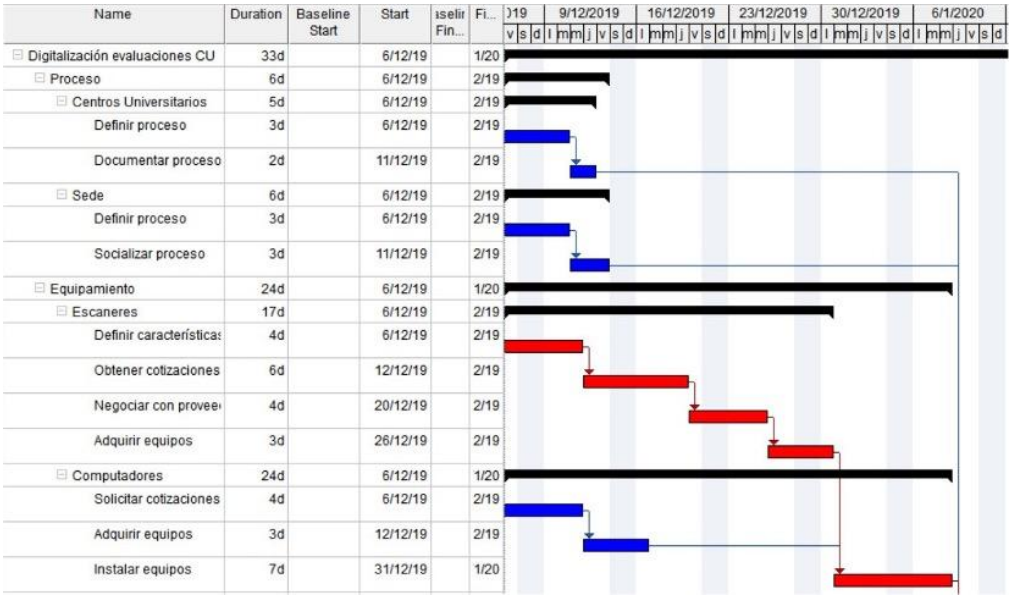
Es un mecanismo que permite mapear las tareas que deben ejecutarse a lo largo del tiempo, cada tarea tiene un conjunto de atributos tales como fecha de inicio, fecha de fin, duración, recursos asignados y la dependencia con otras tareas.

Las notaciones más utilizadas para representar los cronogramas son el diagrama de Gantt y los diagramas Pert. El diagrama de Gantt es una especie de diagrama de barras que en sentido vertical representa la lista de actividades y en sentido horizontal se encuentran las unidades de tiempo y las barras relacionan las actividades con el tiempo en el que deben desarrollarse. En la figura 11 se puede apreciar un diagrama de Gantt utilizado para un proyecto de implementación de escáneres en los centros universitarios de la UTPL y permite digitalizar las hojas de respuesta dadas por los estudiantes para ser enviados electrónicamente y calificados en la sede central.

Ambos diagramas fueron desarrollados utilizando la herramienta WBS Schedule Pro.

Figura 11

Diagrama de Gantt para un proyecto de implementación de escáneres en centros universitarios digitalizar hojas de respuesta de evaluaciones de la UTPL

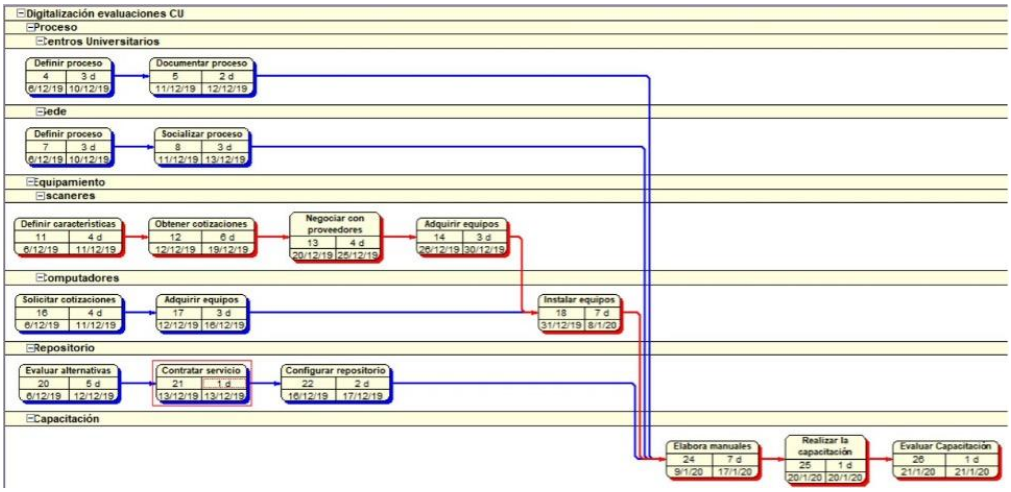


Nota. Abad, M., 2024.

En la figura 12 se aprecia el mismo diagrama representado en diagrama PERT. Cabe resaltar que, si bien ambos diagramas muestran la misma información, cada uno de ellos ofrece una perspectiva diferente del cronograma, el diagrama Pert permite resaltar el tiempo de manera cronológica y el diagrama Pert se enfoca en las dependencias entre actividades, lo que se visualiza en color rojo es lo que se conoce como ruta crítica, esta ruta crítica se usa para establecer la duración del proyecto porque la secuencia de actividades con una duración más larga y, por tanto, las actividades que la conforman no se pueden retrasar.

Figura 12

Diagrama PERT para un proyecto de implementación de escáneres en centros universitarios digitalizar hojas de respuesta de evaluaciones de la UTPL



Nota. *Abad, M., 2024.*

Para profundizar en el tema, revise el apartado 23.3 de su **texto básico** en donde encontrará información sobre cómo desarrollar el cronograma, y además encontrará información sobre cómo se realiza la planificación en un enfoque ágil.

Usted puede descargar *software* para ayudarle en el desarrollo de estos diagramas, algunos son gratuitos como Project Libre, y el caso de WBS Schedule Pro, puede obtener una licencia demostrativa por 30 días o adquirirlo.

En la presente unidad se ha desarrollado una explicación bastante elemental de lo que es la gestión de un proyecto de *software*, con el propósito de que se entienda que el desarrollo de *software* desde el punto de vista profesional no incluye solamente las actividades de programación de una sola persona, sino que implican el trabajo de un equipo de personas cumpliendo diferentes roles y que se sujetan a un cronograma para cumplir con un alcance en un tiempo determinado.

Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Revise el siguiente video sobre la gestión de proyectos de software Ingeniería del software - Gestión de proyectos - Raquel Martínez España.
2. La presente unidad es de tipo informativo, en la cual se ha estudiado algunos conceptos y características de los proyectos de A continuación, le invito a responder las preguntas de la siguiente autoevaluación.

Autoevaluación 3

En las siguientes preguntas escoja la opción correcta:

1. De acuerdo a las definiciones dadas, ¿Cuál de las siguientes alternativas es un proyecto?
    a. Elaborar el documento de requerimientos del sistema.

b. Organizar el equipo de trabajo.
c. Desarrollar una aplicación de ventas para un supermercado.
d. Contratar personal.

2. A las personas que participan en un proyecto se les conoce como:
   a. Participantes.
   b. Equipo del proyecto.
   c. Interesados.
   d. Usuarios.
3. Cuando nos referimos al alcance del proyecto, ¿cuál de las siguientes alternativas es correcta?
   a. En algunas ocasiones el alcance del proyecto incluye al alcance del producto.
   b. El alcance del proyecto es el tiempo del que dispone el equipo de trabajo para completarlo.
   c. El alcance del proyecto es el conjunto de características que debe satisfacer.
   d. El alcance del proyecto describe la estructura del equipo de trabajo.
4. ¿Cuál de las siguientes alternativas es cierta con relación a los interesados del proyecto?
   a. En sentido general, los interesados forman parte del equipo del proyecto.
   b. Los interesados tienen influencia en el proyecto o en su resultado.
   c. Se identifica como interesados a las personas que no se ven afectadas por el proyecto o por su producto resultante.
   d. Los interesados solo tienen influencia al inicio del proyecto.
5. ¿Cuál de los siguientes productos de trabajo puede ser considerado como un entregable?
   a. Modelo de datos.
   b. Diagrama de casos de uso.
   c. Informe de pruebas del software.
   d. Manual de usuario.

6. ¿En cuál de las etapas de un proyecto de software, se intenta comprender los subsistemas que conformarán el sistema?
    a. En la etapa de definición.
    b. En la etapa de ejecución.
    c. En la etapa de inicio.
    d. En la etapa de terminación.
7. ¿De los roles definidos en un proyecto, un tester a qué tipo de rol pertenece?
    a. Roles de gestión.
    b. Roles de desarrollo y cruzado.
    c. Roles de desarrollo.
    d. Roles de consulta.
8. Si al visualizar un cronograma usted encuentra una secuencia de actividades en color rojo, esta representa:
    a. La duración del proyecto.
    b. Las actividades que carecen de recursos.
    c. La ruta crítica.
    d. Las actividades que tienen recursos sobre asignados.
9. Un diagrama de Gantt representa el plan completo de trabajo, pero su diseño permite visualizar mejor:
    a. La asignación de actividades en el tiempo.
    b. Las dependencias entre actividades.
    c. El peso de cada actividad.
    d. Los recursos asignados a cada actividad.
10. ¿En cuál de los diagramas usados para representar el cronograma se evidencia con mayor claridad las dependencias entre actividades?
    a. Diagrama de Gantt.
    b. Diagrama de barras.
    c. Diagrama de actividades.
    d. Diagrama Pert.

Ir al solucionario

Resultado de aprendizaje 4:

Identifica una metodología de desarrollo acorde a las características de un proyecto de *software*.

Para alcanzar el resultado de aprendizaje 4, los estudiantes aprenderán a elegir la metodología de desarrollo más adecuada para un proyecto de *software*, optimizando así el proceso de desarrollo. Para ello, se abarcará el análisis de requerimientos funcionales y no funcionales, la elaboración del documento de requerimientos y el proceso de ingeniería de requerimientos, que incluye la adquisición, análisis y validación de los mismos.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 5

Unidad 4. Requerimientos

En la presente unidad vamos a estudiar los requerimientos de *software*, cómo identificarlos, cómo clasificarlos, cómo documentarlos y cómo utilizarlos para que formen parte del proyecto de desarrollo de *software*.

Los recursos que utilizaremos para el estudio de esta unidad serán fundamentalmente Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL. La unidad 4. Requerimientos. En ella encontrará orientaciones de estudio y actividades de autoevaluación. También trabajará con el **texto básico** Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación.

Para el estudio del contenido vamos a utilizar como recurso principal la guía didáctica de Fundamentos de ingeniería de *software*, la cual le llevará a los recursos necesarios para realizar su proceso de enseñanza aprendizaje.

Para esta unidad, es necesario que usted tenga conocimientos sobre modelado de *software* con UML, este conocimiento ustedes lo adquirieron en la asignatura de Modelado de sistemas, sin embargo, si quiere reforzar o aprender puede usar la unidad 3 de la guía didáctica (Abad, 2020).

4.1. Requerimientos funcionales y no funcionales

En la sección 1.2.4 ya definimos lo que son los requerimientos y se revisaron algunos ejemplos de requerimientos funcionales y no funcionales, en esta sección vamos a profundizar en el concepto, puesto que los requerimientos son la base para iniciar cualquier desarrollo de *software*.

En este momento remítase a la guía didáctica de Fundamentos de ingeniería de *software* y estudie el apartado 4.1, siga todas las instrucciones allí dadas.

Luego de la lectura, realice algunas acotaciones respecto de los Requerimientos de Usuario y los requerimientos del sistema, los Requerimientos de Usuario son más generales, por tanto, describen lo que el usuario espera hacer con el sistema; los requerimientos del sistema son más específicos, por ello describen los servicios que deberá tener el sistema para atender a esas necesidades del usuario.

Esto se evidencia con claridad en los ejemplos de la tabla 5 especificados en la guía didáctica, en el primer ejemplo se especifica que “permitirá al estudiante consultar su expediente académico a través del dispositivo móvil previo método de autentificación”, ello deriva, por tanto, lo que el sistema debe hacer para poder cumplir con ese requerimiento de usuario, en este caso sería: 1. Solicitar al estudiante un usuario y una clave para autenticar su ingreso, 2.

Generar un reporte con todas las asignaturas que el estudiante se ha matriculado, sin considerar las asignaturas actuales que aún está cursando, 3. Posibilidad de descargar el expediente académico en un formato de archivo PDF.

Luego, la clasificación de los requerimientos del *software* en requerimientos funcionales y no funcionales es esencial para las siguientes etapas de desarrollo, como se había indicado en la sección 1.2.4 los requerimientos no funcionales son los que establecen los criterios de calidad del sistema y tienen una gran incidencia en el diseño de la arquitectura del sistema. Si analiza la figura 4.3 de su **texto básico** encontrará que los requerimientos no funcionales también tienen sus categorías denominadas: requerimientos del producto, requerimientos de la organización y requerimientos externos.

Un aspecto muy importante que debe considerarse al momento de trabajar con requerimientos no funcionales es el poder establecer métricas, las cuales se usarán para evaluar la calidad del sistema durante las pruebas.

4.2. El documento de requerimientos de software

Los requerimientos de *software* deben ser documentados, ya que son el principal insumo del desarrollo de *software*, cualquiera de las metodologías estudiadas utiliza los requerimientos y dependiendo de si es ágil o tradicional se utilizarán diferentes formatos para documentarlos.

Hay que tener en cuenta que el documento de requerimientos no es útil solo para los desarrolladores, sino que sirve también para los usuarios, administradores y *testers*.

Para continuar con este tema, remítase por favor al apartado 4.2 de la guía didáctica y realice las actividades allí planteadas.

Muy importante en este tema es analizar el estándar IEEE-830, y poner atención en las formas de especificar los requerimientos de *software*, que van desde expresarlos en lenguaje natural y explicarlos utilizando lenguajes estructurados, cada una de estas formas tiene sus ventajas y desventajas.

Resultado de aprendizaje 4:

Identifica una metodología de desarrollo acorde a las características de un proyecto de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 6

Unidad 4. Requerimientos

4.3. Proceso de ingeniería de requerimientos

El trabajo con los requerimientos va más allá de simplemente recolectarlos y documentarlos, tanto es así que existe un proceso denominado Ingeniería de Requerimientos, que comprende cuatro actividades a nivel general que son: recopilación, especificación, validación y gestión de requisitos.

Para abordar este tema, estudie la sección 4.3 de la guía didáctica, donde explica con detalles el proceso de Ingeniería de Requerimientos.

4.4. Adquisición y análisis de requerimientos

La captura de requerimientos es un proceso que amerita mucho cuidado para los desarrolladores, quienes deben utilizar algunas técnicas y herramientas para capturarlos y en función del modelo de procesos escogido, se capturarán en una sola etapa o se realizará durante las iteraciones. Esto no cambia los instrumentos que se pueden usar.

Ahora estudie el apartado 4.4 de la guía didáctica, cumpliendo con las actividades allí planteadas.

Tanto en la guía didáctica como en el **texto básico** se mencionan varias herramientas para representar los requerimientos tales como diagramas de contexto, mapas de procesos, escenarios, casos de uso, etc. Para nuestro propósito trabajaremos directamente con la especificación a partir de escenarios y casos de uso, debido al valor que aportan para el proceso de Ingeniería de S*oftware* Orientada a Objetos.

Para conocer lo que es y cómo especificar casos de uso, puede visitar la página *web* donde encontrará información muy importante sobre los casos de uso.

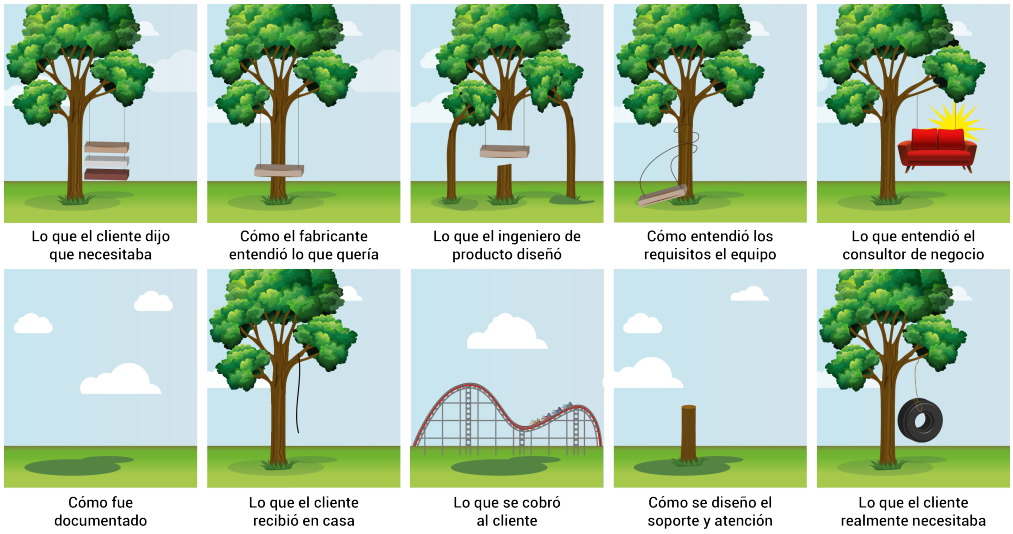
4.5. Validación de requerimientos

No todos los requerimientos se han obtenido pueden tener el sustento necesario, o pueden reflejar las necesidades reales del sistema, por ello es necesario realizar el proceso de validación.

Un dilema común que suele haber entre los ingenieros de *software* es el identificar correctamente lo que el cliente necesita, que muchas de las veces difieren de lo que el cliente dice que necesita. En la figura 13 se aprecia brevemente este dilema, el cual refleja varios actores del proceso de desarrollo fallando en la comprensión de los requerimientos.

Figura 13

Dilema entre lo que el cliente quiere y lo el cliente necesita



Nota. Tomado de Ingenieria en software [Ilustración], por LEVQ, 2013, ingenieriasoftware, CC BY 4.0.

El problema es identificar lo que el cliente realmente necesita, lo que le permitirá resolver los problemas de negocio y así evitar cosas como: reproducir los problemas en el nuevo sistema, o agregar características innecesarias.

Con esto en mente, estudie el apartado 4.5 de la guía didáctica, en el cual se explican algunos mecanismos y técnicas de validación de requerimientos.

Actividades de aprendizaje recomendadas

Para culminar esta unidad, le invito a desarrollar las siguientes actividades:

1. Desarrolle los ejercicios 4.2, 4.3, 4.4, 4.7 del **texto básico**.

2. Revise los siguientes videos, para profundizar en la temática de especificación de requerimientos, especificación de requisitos UCAM y especificación de requisitos UNLa.
3. Desarrolle la autoevaluación 4 para comprobar sus conocimientos.

Autoevaluación 4

En las siguientes preguntas escoja la opción correcta:

1. Cuando se han definido requisitos que no reflejan las necesidades reales de los clientes, se debe a:
    a. Documentos mal redactados.
    b. La metodología de desarrollo no es la apropiada.
    c. No se ha recolectado la información necesaria.
2. A las descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software, se conoce como requerimiento de:
    a. Usuario.
    b. Sistema.
    c. Negocio.
3. Los requerimientos no funcionales se les puede clasificar en:
    a. Usuario y sistema.
    b. Producto y organización.
    c. Producto, organización y externos.
4. A las especificaciones de los servicios y restricciones que el sistema debe considerar, se conoce como requerimiento:
    a. Funcional de usuario.
    b. Funcional de sistema.
    c. No funcionales.
5. Aquellos requerimientos que especifican o restringen el comportamiento del software, se conocen como requerimientos de:
    a. Producto.
    b. Organización.
    c. Externos.

6. El enunciado “El sistema debe ser capaz de operar adecuadamente con hasta 100.000 usuarios con sesiones concurrentes”, es un ejemplo de requerimiento de:
    a. Producto.
    b. Organización.
    c. Externos.
7. La prueba de software se gestionará con la ayuda de una herramienta de testing. Este caso se considera un requerimiento de:
    a. Producto.
    b. Organización.
    c. Externos.
8. Las páginas web a ser desarrolladas deben cumplir con la ley de tratamiento en condiciones de igualdad para personas con discapacidad, es un caso de requerimiento:
    a. Producto.
    b. Organización.
    c. Externos.
9. El proceso de interactuar con los participantes del sistema, le permite:
    a. Descubrir requerimientos.
    b. Priorizar requerimientos.
    c. Clasificar requerimientos.
10. La técnica de validación de requerimientos, que permite mostrar un modelo ejecutable del sistema en cuestión a los usuarios finales y clientes, se conoce como:
    a. Revisión de requerimientos.
    b. Creación de prototipos.
    c. Generación de casos de prueba.

Ir al solucionario

Resultados de aprendizaje 1 y 2:

- Conoce las principales áreas de conocimiento de la Ingeniería de *Software*.
- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.

Para alcanzar los resultados de aprendizaje planteados, los estudiantes adquirirán un conocimiento integral de la Ingeniería de *Software*, incluyendo sus principales áreas y conceptos fundamentales. Además, aprenderán a identificar y comprender las fases del ciclo de vida del desarrollo de sistemas, permitiéndoles aplicar estos conocimientos de manera efectiva en la gestión y ejecución de proyectos de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 7

Este tiempo debe utilizarlo en prepararse para las evaluaciones presenciales, para ello le recomiendo que realice una revisión de las actividades de aprendizaje de las unidades 1 y 2, asegurándose de que comprende las temáticas tratadas, los videos que se recomiendan al final de cada unidad le serán de mucha utilidad.

Como estrategia de estudio, le recomiendo elaborar mapas mentales, si no ha trabajado con ellos puede revisar el siguiente video ¿Qué son los mapas mentales y cómo se crean?, y puede encontrar algún software que ayude con elaborarlos y gestionarlos con mayor facilidad.

Resultados de aprendizaje 1 a 4:

- Conoce las principales áreas de conocimiento de la Ingeniería de *Software*.
- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.
- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de *software*
- Identifica una metodología de desarrollo acorde a las características de un proyecto de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 8

Actividades finales del bimestre

Para esta semana, se le recomienda repasar las unidades 3 y 4, tenga en cuenta que la unidad 3 tiene mucho contenido teórico, por lo que es recomendable hacerse un mapa mental del mismo.

La unidad 4 es más compleja y necesaria para el segundo bimestre, por lo que es necesario que ponga más atención y repase los ejercicios sobre requerimientos, recuerde que se han colocado algunos videos que puede repasar.

En caso de dudas, utilice el recurso de la tutoría, no se encuentra solo, su tutor le acompañará a lo largo de este proceso.

Segundo bimestre

Resultados de aprendizaje 1 y 2:

- Conoce las principales áreas de conocimiento de la Ingeniería de *Software*.
- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.

Para alcanzar los resultados de aprendizaje planteados, los estudiantes explorarán las principales áreas de conocimiento en Ingeniería de *Software* y aprenderán a identificar las fases del ciclo de vida del desarrollo de sistemas. Se enfocarán en el análisis, comprendiendo sus conceptos, actividades y gestión, lo que les permitirá aplicar estos conocimientos en la planificación y ejecución de proyectos de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.

Semana 9

El modelo de análisis es un paso intermedio para llegar del modelo de requerimientos al diseño, y del diseño a la implementación, por tanto, en este bimestre vamos a trabajar en realizar este proceso, recuerde que este contenido no está sujeto a ninguna metodología de desarrollo, lo que se estudia aquí debe aplicarse en cualquiera de ellas bien sea con un enfoque predictivo o con un enfoque ágil.

Desde el punto de vista de ingeniería, estamos estudiando el método de construcción de las aplicaciones, la metodología de desarrollo de *software* organiza las actividades del método con su enfoque particular en función de las restricciones del proyecto.

Como recurso de aprendizaje que utilizará será: Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL, concretamente la unidad 5.

Además, para poder realizar las prácticas, se utilizará un laboratorio son *software* especializado, su tutor le dará las instrucciones necesarias para trabajar con el mismo.

Para realizar el análisis orientado a objetos, se requiere el artefacto de especificación de los casos de uso, y para poder trabajar con el modelo de análisis es necesario contar con esas especificaciones.

Comience realizando una lectura de la introducción a la unidad 5 de la guía didáctica, se hace una explicación general del tema y una estrategia para el trabajo, tal como menciona la guía puede profundizar en el modelado revisando el texto: The Unified Modeling Languaje.

Adicionalmente, le recomiendo utilizar la siguiente lista de videos para refrescar o aprender temas de modelos con UML.

- Modelado de casos de uso UPV.
- Modelado de clases con UML de la UCAM.

Además, le animo a revisar el libro de Penalvo, F. J. G., Martin, S. B., González, M. A. C. (2008,). Ingeniería del *software*. Retrieved March 27, 2020, from OCW-USAL Web site.

Unidad 5. Análisis

5.1. Conceptos del análisis

Bien, si ya pudo revisar los videos y descargó el texto indicado, comencemos revisando los conceptos del análisis.

Realice un estudio previo del OCW de la Universidad de Salamanca, el apartado "Análisis orientado a objetos" descargando el documento tema 4: Análisis orientado a objetos.

Ahora estudie el apartado 5.1 de la guía didáctica, y realice las lecturas y las actividades indicadas.

Los conceptos importantes en este apartado son el modelo de clases, los modelos de interacción y lo más significativo la clasificación de los diferentes tipos de objetos denominados como entidad, borde y control.

El ejercicio del cajero automático indicado en la guía didáctica, resulta fácil identificar los diferentes tipos de objetos, debido a la presencia de elementos físicos que simplifican la clasificación de los diferentes tipos de objetos, sin embargo, en los modelos de *software* que no tienen una alta dependencia de modelos físicos, esto no siempre es evidente y por ello es necesario trabajar con las heurísticas, que se estudian en el siguiente apartado.

5.2. Actividades del análisis

Para estudiar este apartado, le invito a estudiar la sección 5.1 Actividades del análisis de la guía didáctica.

En principio el desarrollo de estas actividades es importante desarrollarlo paso a paso, y sobre todo resulta fundamental aplicar las heurísticas recomendadas para cada uno de los tipos de objetos, como habrá notado si no cuenta con la especificación de los casos de uso, difícilmente podrá desarrollar estas actividades, esto ya lo habíamos mencionado en la unidad 4.

La identificación de los diferentes tipos de objetos es el primer paso, luego es necesario crear diagramas de secuencia que nos permitan finalmente construir un diagrama de clases del análisis completo con sus respectivos atributos y relaciones.

El modelo de análisis también incluye el diagrama de estados, también conocido como máquina de estados.

Finalmente, es necesario validar el modelo de análisis, para ello es necesario chequear una serie de características establecidas en la tabla 13 de la guía didáctica.

5.3. Gestión del análisis

Para completar el modelo de análisis es necesario gestionarlo, ello implica documentar el modelo de análisis y asignar responsabilidades, para este apartado sírvase estudiar el apartado 5.3 de la guía didáctica.

Estimado estudiante, estas actividades tienen un nivel de complejidad mayor, por ello asegúrese de haber comprendido la temática, los ejercicios y sobre todo de haber desarrollado con éxito la actividad propuesta antes de pasar al siguiente tema.

Cuenta con la ayuda de su tutor y puede contactarlo por los medios indicados previamente.

Actividades de aprendizaje recomendadas

Es momento de aplicar sus conocimientos a través de las actividades que se han planteado a continuación:

1. Considere el caso de un sistema para controlar el ingreso de personas a un evento académico, el sistema comprende 4 casos de

uso como registrar participantes, registrar ingreso al evento, emitir certificados y calificar el evento. Desarrolle lo siguiente:

- Diagrama de casos de uso considerando los actores siguientes: operador, que es quien registra a los participantes, y emite los certificados; controlador, que es quien registra el ingreso de los participantes a cada uno de los eventos, y el participante que califica el evento.
- Especificación de casos de uso.
- Modelo de análisis que incluye: aplicar las heurísticas para identificar los objetos del análisis, elaborar diagramas de secuencia y de estados para refinar las características de los objetos y elaborar el diagrama de clases del análisis.

Nota: por favor, complete la actividad en un cuaderno o documento Word.

2. Estimado estudiante, le invito a desarrollar la siguiente autoevaluación de la unidad 5.

Autoevaluación 5

En las siguientes preguntas escoja la opción correcta:

1. El modelo de análisis se compone de tres modelos individuales, ¿A cuál de estos modelos pertenecen los casos de uso?
    a. Modelo de objetos.
    b. Modelo funcional.
    c. Modelo dinámico.
2. En el modelo dinámico del análisis, los diagramas que representan el comportamiento de un objeto individual son:
    a. Diagramas de secuencia.
    b. Diagramas de clases.
    c. Diagramas de estados.

3. De acuerdo a las descripciones dadas respecto de los diferentes estereotipos de los objetos identificados durante el análisis, los objetos de control cumplen la siguiente función:
    a. Se encargan de la realización de los casos de uso.
    b. Permiten la comunicación con el actor.
    c. Representan la información persistente del sistema.
4. Considere un sistema de archivos con una interfaz gráfica de usuario, tal como el Mac Finder o Windows Explorer. Los siguientes objetos se identificaron desde un caso de uso, describiendo cómo copiar un archivo desde una unidad de disco: archivo, ícono, basurero, carpeta, disco, puntero. Seleccione la alternativa que clasifique correctamente a los objetos en entidad, cuáles son borde y cuáles son de control.
    a. Entidad: ícono, carpeta; Borde: basurero, puntero; Control: disco, archivo.
    b. Entidad: archivo, disco; Borde: carpeta, basurero; Control: puntero.
    c. Entidad: archivo, carpeta, disco; Borde: ícono, puntero, basurero; Control: ninguno.
5. Una característica importante del modelado orientado a objetos, es que asocia entidades del mundo real, con clases del software. En la siguiente lista de objetos para un sistema de reloj satelital capaz de autoajustar la hora en función de la localización geográfica (zona horaria), identifique aquellos que corresponden a clases del software. Seleccione todas las que apliquen.
    a. LocalizadorGPS
    b. HoraUniversal
    c. ZonaHoraria
    d. BasedeDatosZonaHoraria
    e. Ubicación
    f. IdentidadUsuario
6. La generalización y la especialización dan como resultado que una clase padre o superclase herede sus características a una o más

clases hijas, sin embargo, conceptualmente son cosas diferentes ¿Cuál es la diferencia entre generalización y especialización?

a. La generalización crea conceptos abstractos a partir de conceptos de bajo nivel y la especialización identifica conceptos específicos a partir de conceptos generales.
b. La generalización trabaja de lo general a lo particular para crear relaciones jerárquicas y la especialización opera desde lo general a lo particular.
c. Son sinónimos y no hay diferencia.

7. De acuerdo a la heurística de Abbot, determine qué tipo de relación debería generarse a partir de la siguiente descripción: “La estación de bomberos está conformada por un jefe y muchas brigadas, las cuales a su vez conforman por bomberos”

a. Agregación por composición.
b. Agregación compartida.
c. Asociación.

8. En un sistema de cajero automático, se desarrolla un diagrama de secuencia para un escenario del caso de uso “Retirar Dinero”, se establece que el actor Cliente <<entidad>>, envía el mensaje ObtenerDinero (monto) al objeto RetirarFondos <<control>>, luego RetirarFondos <<control>> responde al actor Cliente <<entidad>> “no dispone de ese monto”. Analice el contexto y seleccione la alternativa que describa un juicio válido respecto del escenario.

a. La relación es incorrecta, ya que no se puede comunicar directamente a un actor con un objeto de control.
b. La relación es correcta, debido a que no se atenta ninguna de las normas dadas.
c. La relación está incompleta, ya que faltan elementos clave.

9. Durante la validación del modelo de análisis, se plantea que el mismo debe estabilizarse para dar por terminada la etapa. ¿Cuál de las siguientes alternativas significa estabilizarse?

a. Ya no hay más cambios.
b. Se ha dispuesto que ya no se sigan haciendo cambios.
c. El número de cambios se han minimizado.

10. ¿Cuál de los siguientes roles se hace cargo de integrar el modelo de casos de uso con el modelo de objetos?
    a. Analista.
    b. Arquitecto.
    c. Programador.

Ir al solucionario

Resultados de aprendizaje 3 y 4:

- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de *software*.
- Identifica una metodología de desarrollo acorde a las características de un proyecto de *software*.

Para alcanzar los resultados de aprendizaje planteados, los estudiantes aprenderán a identificar el ciclo de vida y la metodología de desarrollo más adecuada para un proyecto de *software*. Además, se enfocarán en el diseño arquitectónico, comprendiendo las decisiones clave, conceptos fundamentales, patrones arquitectónicos y arquitecturas de aplicación. Este conocimiento les permitirá seleccionar y aplicar el enfoque más efectivo para el desarrollo de *software* según las características del proyecto.

Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.

Semana 10

El diseño del sistema es el modelo más importante para la construcción de aplicaciones, para construirlo se debe partir del modelo de análisis y luego seleccionar un estilo arquitectónico que permita satisfacer los requerimientos no funcionales, en esta unidad se revisarán algunos conceptos de arquitectura de *software* y se revisarán los principales patrones arquitectónicos y las arquitecturas de aplicación más comunes.

Como recurso de aprendizaje que utilizará será: Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja- Ecuador. Editorial UTPL, concretamente la unidad 6. Y el **texto básico** Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación.

Unidad 6. Diseño arquitectónico

Comience revisando la parte introductoria a la unidad 6 de la guía didáctica, en ella se le pide realizar la lectura del **texto básico** y se le ofrece algunas pautas de lo que es el diseño arquitectónico.

Luego de la lectura es importante intentar responder a las preguntas planteadas, ya que permitirán orientar el propósito del diseño arquitectónico del sistema.

6.1. Decisiones en el diseño arquitectónico

Como se había mencionado anteriormente, las decisiones de diseño arquitectónico se basan en los criterios de calidad y requerimientos no funcionales del sistema, de cómo se utilice estos requerimientos dependen en gran medida la calidad del sistema.

Estudie el apartado 6.1 de la guía didáctica para visualizar el proceso de diseño y cómo se relacionan las actividades del diseño con los diferentes artefactos, como los requerimientos no funcionales y los artefactos resultantes del análisis. Esto lo evidencia en la figura 25 de la guía didáctica.

Las decisiones de diseño también tendrán impacto en el rendimiento, en la seguridad, en la mantenibilidad y otros atributos de calidad, de hecho, los atributos de calidad son clave al momento de tomar este tipo de decisiones.

Vale tener en cuenta los estándares de calidad del *software* como producto, en este caso un estándar ISO es la norma ISO9126.

6.2. Conceptos de diseño y arquitectura del sistema

Para abordar el tema del diseño, es importante primero entender algunos conceptos, por ello se recomienda poner especial cuidado en el estudio de estos temas y de ser posible profundizarlos utilizando otros recursos.

Ahora por favor revise el apartado 6.2 Conceptos de diseño de la guía didáctica y trate de asimilar estos conceptos, recuerde desarrollar las actividades recomendadas y acudir a su tutor en caso de que se le presente dudas.



Resultados de aprendizaje 3 y 4:

- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de *software*.
- Identifica una metodología de desarrollo acorde a las características de un proyecto de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 11

Unidad 6. Diseño arquitectónico

6.3. Patrones arquitectónicos

Un patrón o estilo arquitectónico es “un conjunto de principios generales que proporcionan marco de trabajo abstracto para una familia de sistemas” (Microsoft Corporation, 2009), estos principios son soluciones a problemas comunes de diseño de *software*, por ello es importante conocerlos y aplicarlos. En la siguiente infografía se resumen algunos patrones arquitectónicos de uso común.

Estilos arquitectónicos de uso común

Asumiendo las instrucciones dadas en la guía didáctica, sírvase estudiar el apartado 6.3 del **texto básico** e, intente responder las preguntas planteadas en la sección 6.3 de la guía didáctica.

En los últimos años, se ha dado un gran auge en el uso de un nuevo estilo arquitectónico denominado Arquitectura Orientada a Microservicios, en el siguiente video, encontrará algunas pautas para entender qué es y cómo funciona: Video microservicios.

6.4. Arquitecturas de aplicación

Para el estudio de este apartado remítase a la sección 6.4 de la guía didáctica y, lleve a cabo las actividades planteadas.

Tenga en cuenta que puede profundizar en este tema accediendo al enlace propuesto.

También puede revisar el artículo de Garlan y Shaw, que incluye diferentes Estilos arquitectónicos y casos de estudio.

Actividades de aprendizaje recomendadas

Reforcemos el aprendizaje resolviendo las siguientes actividades:

1. Revisar el video Microservicios | ¿Qué son los microservicios?, introducción a microservicios para conocer un poco más sobre la arquitectura orientada a microservicios.
2. Desarrolle la siguiente autoevaluación de la unidad 6 para comprobar sus conocimientos.

Autoevaluación 6

En las siguientes preguntas escoja la opción correcta:

1. El autor del texto básico, propone que la arquitectura del software se puede evidenciar a dos niveles, a nivel pequeño y arquitectura en grande, por otro lado, si contrasta estas definiciones con los patrones arquitectónicos, ¿a qué nivel se mueve el modelo Vista – Controlador?
    a. Pequeño.
    b. Grande.
    c. Es independiente.

2. Cuando se diseña la arquitectura del software, es necesario considerar tanto requerimientos funcionales como no funcionales, los cuales se implementan tanto con componentes individuales como en esquemas arquitectónicos. ¿En cuál de los siguientes componentes se implementan los requerimientos funcionales?
    a. A nivel de arquitectura.
    b. En componentes externos.
    c. En componentes individuales.
3. Cuando se habla de las ventajas de documentar la arquitectura se plantea el hecho de que favorece la comunicación con los participantes ¿Por qué motivo sucede esto?
    a. Porque es una presentación de alto nivel del sistema.
    b. Porque se elabora es una etapa temprana del desarrollo.
    c. Porque muestra cómo se organiza el sistema.
4. ¿Por qué el modelo de análisis es insuficiente para poder pasar a la implementación del sistema?
    a. Carece de información de la estructura interna del sistema.
    b. Las descripciones de los casos de uso no son lo suficientemente detalladas.
    c. Falta complementar los diagramas de clases.
5. ¿Cuál de las siguientes características debe implementarse en la arquitectura del sistema para garantizar la disponibilidad?
    a. Activos críticos del sistema protegidos.
    b. Operaciones críticas del sistema implementadas ubicadas en algún componente individual.
    c. Incluir componentes redundantes.
6. ¿Por qué motivo no es posible que una aplicación cumpla con todos los atributos de calidad?
    a. Porque algunos pueden entrar en conflicto con otros.
    b. Puedes resultar demasiado costoso.
    c. Puede tomar demasiado tiempo implementarlo.
7. Según el modelo arquitectónico propuesto por Krutchen 4+1, ¿en cuál de las vistas se aprecian las clases del sistema?
    a. Vista de desarrollo.

b. Vista de proceso.
c. Vista lógica.

8. ¿Qué lenguajes se utilizan específicamente para describir la arquitectura del sistema?
    a. UML.
    b. ADL.
    c. C.
9. El uso de metodologías ágiles se ha popularizado y se utilizan en casi cualquier ámbito del quehacer humano, este movimiento inició en la industria del software, ¿qué sucede con la arquitectura desde el punto de vista ágil para el desarrollo de software?
    a. Ya no se necesita, puesto que el sistema evoluciona conforme se avanza en los requerimientos.
    b. Debe ser más robusta que en las no ágiles.
    c. El nivel de especificación de la arquitectura no depende de la metodología sino de la complejidad del sistema.
10. Al descomponer un sistema en subsistemas, surgen dos conceptos esenciales: la cohesión y el acoplamiento y pueden entrar en conflicto dada la dependencia entre ellos ¿Cuál es la recomendación al respecto?
    a. Priorizar la cohesión.
    b. Aplicar la heurística 7 ± 2.
    c. Minimizar el acoplamiento.

Ir al solucionario

Resultados de aprendizaje 2 y 3:

- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.
- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de *software*.

Para alcanzar los resultados de aprendizaje planteados, los estudiantes desarrollarán una comprensión profunda de las fases del ciclo de vida del desarrollo de sistemas y aprenderán a elegir el ciclo de vida más adecuado para cada proyecto de *software*. Se enfocarán en el diseño e implementación, incluyendo el diseño orientado a objetos con UML, el uso de patrones de diseño y la resolución de conflictos durante la implementación. Además, se estudiarán los conceptos esenciales de las pruebas de *software*, las pruebas de desarrollo y la gestión de las actividades de prueba. Este conocimiento les permitirá aplicar métodos efectivos para diseñar, implementar y asegurar la calidad en sus proyectos de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Recuerde revisar de manera paralela los contenidos con las actividades de aprendizaje recomendadas y actividades de aprendizaje evaluadas.

Semana 12

Una vez que ha estudiado los estilos arquitectónicos, el diseño del sistema se convierte en una actividad más técnica que busca crear modelos que permitan la implementación de aplicaciones que cumplan con las especificaciones tanto funcionales como no funcionales. En esta unidad se estudiará el Diseño Orientado a Objetos.

Como recurso de aprendizaje que utilizará la unidad 7 del recurso Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL. Y el **texto básico** Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación.

Unidad 7. Diseño e implementación

En esta unidad vamos a revisar las actividades de modelado del Diseño Orientado a Objetos, el propósito de esta unidad es crear el diseño de los subsistemas que deberán implementarse y construir los modelos de implantación.

Realice la lectura introductoria al apartado 7 de la guía didáctica.

7.1. Diseño orientado a objetos con UML

Esta etapa del proceso de diseño comienza con el mapeo de subsistemas a plataformas de *hardware* y *software* que se representa en UML con diagramas de despliegue (*deployment diagram*), luego está el mapeo de información persistente, definir políticas de control de acceso, seleccionar el flujo global de control y definir condiciones de operación externas.

Para el **mapeo a plataformas de** *hardware* **y** *software* de acuerdo a (Bruegge & Dutoit, 2010) es necesario plantearse las siguientes preguntas:

- ¿Cuál es la configuración de *hardware* del sistema?
- ¿Qué nodo es responsable de qué unidad?
- ¿Cómo realizar la comunicación entre nodos?
- ¿Qué servicios se realizan utilizando componentes de *software* existentes?
- ¿Cómo se encapsulan estos componentes?

La respuesta a estas preguntas suele derivar en la definición de subsistemas adicionales que permiten mover los datos de un nodo a otro, resolviendo problemas de concurrencia y de confiabilidad.

En el mercado también existen componentes listos para usar que permiten implementar servicios complejos de forma sencilla, entre estos podemos encontrar paquetes de Interfaz de Usuario o sistemas de administración de base de datos que encapsulados debidamente minimizan la dependencia entre componentes.

En cuanto la **gestión de datos,** deberíamos preguntarnos lo siguiente:

- ¿Qué datos deberían ser persistentes?
- ¿Dónde se debe almacenar esta información?
- ¿Cómo será el acceso a la misma?

La persistencia de datos es un cuello de botella en el sistema de muchas formas diferentes, la mayor parte de la funcionalidad del sistema realiza operaciones de creación o manipulación de la información, lo cual implica que el acceso a la información persistente debería ser rápido y sobre todo confiable, de lo contrario el sistema sería lento y la información podría perderse fácilmente. Este problema se resuelve normalmente con un Sistema de Gestión de Base de Datos (DBMS) de los existentes en el mercado como Oracle, SQLServer, MySQL o algún otro.

Las **políticas de control de acceso,** se definen con base en la respuesta a las siguientes preguntas:

- ¿Quién accede a qué información?
- ¿Pueden los controles de acceso cambiar de forma dinámica?
- ¿Cómo se especifica y se implementa el control de acceso?

El control de acceso y la seguridad son grandes preocupaciones del sistema, este debe ser consistente y la política debe ser capaz de definir quién puede o no acceder a cierta información y si debería ser igual para todos los subsistemas.

En lo relacionado con el **flujo de control,** nos podemos plantear las siguientes preguntas:

- ¿En qué secuencia se realizan las operaciones en el sistema?

- ¿El sistema es dirigido por eventos?
- ¿Puede el sistema manejar más de una interacción del usuario a la vez?

La selección del flujo de control tiene un alto impacto en las interfaces de los subsistemas. Si se selecciona un flujo de control dirigido por eventos, los subsistemas deberán proporcionar manejadores de eventos, si en lugar de ello se seleccionan hilos, el subsistema debe garantizar exclusión mutua en secciones críticas.

En cuando a las **condiciones de operación externas (condiciones límite),** las preguntas a plantearse son:

- ¿Cómo se levanta y cómo se baja el sistema?
- ¿Cómo se tratan los casos excepcionales?

El inicio y apagado del sistema representa un alto nivel de complejidad de sistema, especialmente en sistemas distribuidos y tienen mucho impacto en las interfaces de todos los subsistemas.

Ahora es momento de estudiar el apartado 7.1 de la guía didáctica, donde profundizará este tema y podrá revisar información y recursos adicionales.

Resultados de aprendizaje 2 y 3:

- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.
- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 13

Unidad 7. Diseño e implementación

7.2. Patrones de diseño

Los patrones de diseño son soluciones generales para problemas comunes de diseño de aplicaciones, son muchos los patrones que se pueden y se deben usar dependiendo de la situación que estemos resolviendo, para ello es importante estudiarlos y aplicarlos cuando se vea necesario.

Para estudiar este tema, remítase a la guía didáctica apartado 7.2 en ella encontrará referencias para descargar material explicativo de este tipo de patrones.

7.3. Conflictos de implementación

Durante la implementación de un sistema es normal que se presenten diferentes tipos de conflictos, en este apartado se estudia ¿cuáles son y cómo abordarlas?

Para ello lo invito a estudiar el apartado 7.3 de la guía didáctica. Desarrolle todas las actividades planteadas, nuevamente se le recuerda acudir a su tutor en caso de dudas.

Actividades de aprendizaje recomendadas

Para afianzar sus conocimientos sobre las temáticas abordadas en esta unidad, desarrolle las siguientes actividades planteadas:

1. Realice las actividades presentadas en la guía didáctica correspondientes a esta unidad.
2. Luego ponga a prueba su aprendizaje desarrollando los siguientes ejercicios, en caso de dudas no olvide acudir a su tutor.
    a. Considere un sistema que incluya 1 servidor *web* y 2 servidores de base de datos. Ambos servidores de base de datos son idénticos, el primero actúa como servidor principal y el segundo como servidor redundante en caso de que el servidor 1 falle. Los usuarios utilizan un navegador *web* para acceder a los datos mediante el servidor *web*. Además, tienen la opción de usar un cliente propietario que accede a la base de datos directamente. Dibuje un diagrama de despliegue en UML que represente el mapeo de *hardware* y *software*.
    b. Está diseñando las políticas de control de acceso para una tienda minorista basada en la *web*. Los clientes acceden a la tienda a través de la *web,* navegan por la información del producto, ingresan su dirección e información de pago y compran productos. Los proveedores pueden agregar nuevos productos, actualizar información del producto y recibir pedidos. El dueño de la tienda establece los precios minoristas, hace ofertas personalizadas a los clientes en función de sus perfiles de compra y ofrece servicios de *marketing*. Usted debe considerar tres actores: administrador de la tienda, proveedor y cliente. Diseñar una política de

control de acceso para los tres actores. Los clientes pueden ser creados a través de la *web*, mientras que los proveedores son creados por el administrador de la tienda.

3. Desarrolle la siguiente autoevaluación de la unidad 7 para comprobar sus conocimientos.

Autoevaluación 7

En las siguientes preguntas escoja la opción correcta:

1. Una de las preocupaciones del diseño, es la definición de objetivos de diseño. Estos objetivos de diseño ¿Con cuál de los siguientes aspectos se encuentra más relacionado?
    a. Requerimientos funcionales.
    b. Requerimientos no funcionales.
    c. Necesidades de usuario.
2. Cuando se modela el contexto del sistema ¿Cuál de las siguientes alternativas describe en qué consiste el contexto?
    a. Los sistemas externos con los cuales se relaciona el sistema.
    b. Los condicionantes internos y condiciones límite del sistema.
    c. Las restricciones impuestas por el usuario.
3. El propósito del diseño arquitectónico es:
    a. Identificar los casos de uso que serán atendidos por el sistema.
    b. Identificar las clases que resuelven las necesidades del usuario.
    c. Identificar subsistemas que conformarán el sistema.
4. Los modelos de diseño comprenden tres tipos de modelos. ¿A qué categoría corresponden aquellos que representan cómo los objetos individuales cambian en respuesta a eventos?
    a. De subsistema.
    b. De secuencia.
    c. De máquina de estados.

5. ¿En qué tipo de diagramas de UML se representan los dispositivos donde se instalarán los componentes y las relaciones entre ellos?
    a. Diagrama de componentes.
    b. Diagrama de implantación.
    c. Diagrama de actividades.
6. ¿Cuál de los siguientes elementos es un candidato obvio para ser almacenado de manera persistente?
    a. Objetos entidad.
    b. Objetos borde.
    c. Objetos de control.
7. ¿Qué problema pueden tener las bases de datos orientadas a objetos al momento de almacenar la información persistente?
    a. Pueden representar tanto los objetos como sus relaciones.
    b. Son difíciles de implementar.
    c. El rendimiento es bajo.
8. ¿Qué beneficio ofrece la protección con encriptación?
    a. Se necesita desencriptarla para leerla.
    b. Es muy rápida de obtener el código encriptado.
    c. Si alguien logra robarla, no la podrá entender.
9. ¿Cómo ayudan al proceso de diseño los patrones?
    a. Proveen soluciones específicas para problemas generales.
    b. Proveen soluciones generales a problemas particulares.
    c. Ayudan a identificar componentes de software para resolver problemas de diseño.
10. ¿Qué beneficio ofrecen las herramientas de administración de la configuración al trabajo en equipo?
    a. Proveen estándares de codificación.
    b. Almacenan la información del proyecto en un repositorio central.
    c. Evitan conflictos de uso de código por parte de varios programadores.

Ir al solucionario

Resultados de aprendizaje 2 y 3:

- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.
- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de *software*.

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 14

Para concluir el estudio de la presente asignatura, vamos a estudiar lo que son y en qué consisten las pruebas del *software*, las cuales permitirán cerrar el ciclo de desarrollo incorporando procesos de pruebas del *software* para garantizar que el mismo cumple con las especificaciones funcionales y no funcionales.

Como recurso de aprendizaje que utilizará la unidad 8 del recurso Abad, M. (2020). Guía didáctica de Fundamentos de ingeniería de *software*. Loja-Ecuador. Editorial UTPL. Y el **texto básico** Sommerville, I. (2011). Ingeniería de *software*. (9.ª edición). México. Pearson Educación.

Unidad 8. Pruebas del software

En la ingeniería de *software*, para desarrollar productos de calidad que cumplan con las especificaciones, no basta el uso de una buena metodología de desarrollo o un diseño arquitectónico robusto y eficiente, es necesario realizar pruebas de diferente tipo y en diferentes etapas del proceso. En los modelos ágiles se habla de una cultura de calidad, sin embargo, eso no significa solo hacer las cosas bien, sino probarlas más a menudo y la diferencia fundamental entre las metodologías ágiles y las tradicionales es que en las primeras todo el mundo es responsable de la

calidad y en las segundas hay un equipo encargado del aseguramiento de la calidad que son los especialistas en realizar las pruebas y reportar los resultados a los desarrolladores para que realicen las correcciones necesarias.

Independientemente de cuál sea el enfoque, en esta unidad vamos a estudiar los diferentes tipos de pruebas que se pueden aplicar al *software*.

Para comenzar le invito a realizar el estudio de la introducción a la unidad 8 de la guía didáctica, siga las instrucciones dadas.

Un aspecto a resaltar y que en la gran mayoría de proyectos sucede es el planteamiento de Dijstra que menciona que “las pruebas pueden mostrar la presencia de errores, más no su ausencia”, con ello nos hace entender que por exhaustivo que sea el proceso de pruebas, es posible que se encuentren errores aun cuando las pruebas realizadas no los reporten, y muchas de las veces estos errores se producen cuando el producto está en manos del usuario, en un proceso tradicional se trata de un encuentro tardío y que puede resultar costoso al momento de corregir o incluso se puede enfrentar demandas, en el caso de las metodologías ágiles, la probabilidad de encontrarlo de manera inmediata una vez liberado un incremento es muy alta y, por tanto, la corrección impactará mucho menos que en la tradicional.

8.1. Conceptos relacionados con las pruebas

Ahora revisemos algunos conceptos y terminología común de las pruebas.

Estudie el apartado 8.1 de la guía didáctica, en él encontrará mucha de la terminología frecuente en las pruebas que se estudiarán más adelante.

Todos los conceptos estudiados definen una serie de parámetros que se usarán durante los procesos de prueba del *software*, por tanto, es conveniente que les dé una nueva mirada antes de pasar al siguiente apartado.

8.2. Pruebas de desarrollo

Habíamos indicado que las pruebas se realizan a diferentes niveles, unas las realizan los mismos desarrolladores o programadores y otras las ejecutan los especialistas en pruebas, también conocidos como *testers*.

Es momento de estudiar la guía didáctica, remítase al apartado 8.2 para determinar lo que deben hacer los desarrolladores durante su proceso de pruebas.

Las pruebas de desarrollo, por tanto, son las siguientes:

- **Pruebas de unidad:** aplicadas a un módulo individual a ver si cumple con la especificación, pero también es necesario determinar si no existen errores en el código por alguna bifurcación no controlada, por ello recomienda el uso herramientas de prueba automatizadas que permitan probar todos los casos posibles. Al final de la unidad tendrá algunas referencias sobre herramientas para automatización de pruebas.
- **Pruebas de componentes:** en realidad son pruebas aplicadas a subsistemas, son un poco más amplias que las anteriores y buscan la integración entre clases y de interfaces. Debe tener en cuenta la estrategia incremental para la integración, esto suele ser muy costoso de realizar.
- **Pruebas de sistema:** estas pruebas están orientadas a determinar el funcionamiento correcto del sistema y se centran en atributos de calidad como la funcionalidad, el rendimiento. Además, incluyen pruebas de aceptación del cliente y de instalación para asegurarse que el cliente no tendrá inconvenientes.

8.3. Actividades y gestión de las pruebas

El proceso de pruebas suele ser muy formal precisamente para asegurar que todo funciona correctamente, sin embargo, debemos tener en cuenta que los productos de *software* son extremadamente complejos y para efectuar las pruebas es necesario contar con instrumentos como métricas que nos permitan establecer el nivel que se está logrando.

Es momento de tomar la guía didáctica y desarrollar las actividades de aprendizaje de la sección 8.3, asegúrese de comprender el contenido y solicitar el apoyo de su tutor en caso de que lo requiera.

Felicidades, con esto ha concluido su estudio de la asignatura

Actividades de aprendizaje recomendadas

Continuemos con el aprendizaje mediante su participación en las actividades que se describen a continuación:

1. Le invito a revisar el *software* de automatización de pruebas visitando el sitio *Software Testing Help*, allí se listan y se comentan un conjunto de herramientas *open source* para automatización de pruebas.
2. Desarrolle la autoevaluación de la unidad 8. Recuerde que debe desarrollarla por completo y tenga en cuenta que las partes que no resolvió correctamente, son las que debe repasar y en caso de dudas puede acudir a su tutor. Las autoevaluaciones son una estrategia para medir su nivel de comprensión de la asignatura, por tanto, no termine su estudio sin haberlas resuelto.

Autoevaluación 8

En las siguientes preguntas escoja la opción correcta:

1. ¿La diferencia entre verificación y validación es?
    a. No hay diferencia, son sinónimos.
    b. La verificación sirve para determinar si el producto cumple con las expectativas del cliente y la validación si el producto se construyó correctamente.
    c. La verificación sirve para determinar si el producto se construyó correctamente y la validación si el producto cumple con las expectativas del cliente.
2. Según Prowel et al. (1999), más del 90 % de defectos pueden encontrarse realizando inspecciones, sin embargo, en la práctica estas no resultan tan buenas. Señale el motivo por el que pueden verse afectadas estas previsiones.
    a. Las inspecciones no consideran las interfaces entre componentes.
    b. Las inspecciones permiten analizar código, y este no es claro para los responsables de las pruebas.
    c. Suele ser muy complejo reunir un equipo para realizar las inspecciones.
3. Usted lidera un equipo de proyectos para desarrollar software, y en los primeros componentes de software liberados para pruebas, encuentra cientos de errores que son identificados por los equipos de pruebas cada vez más rápidas. Este es un claro indicio de:
    a. Curva normal de detección de errores.
    b. Riesgo de errores en componentes futuros.
    c. La experiencia del equipo de pruebas minimiza los errores.
4. ¿Cuál de las siguientes alternativas expresa el significado de la palabra “fallo” en el contexto de las pruebas del software?
    a. Descripción de los resultados de las pruebas realizadas.
    b. Imperfección del software.
    c. Efecto indeseado en las funciones o prestaciones del sistema.

5. ¿A qué nivel se desarrollan las pruebas que toman un objeto o una operación de un objeto para determinar si cumple con la especificación?
    a. Componentes.
    b. Integración.
    c. Sistema.
6. ¿Cuál es el objetivo de las pruebas de componentes?
    a. Determinar si los componentes cumplen con las especificaciones.
    b. Valorar la usabilidad del componente.
    c. Determinar si las interfaces funcionan correctamente.
7. Cuando se realizan pruebas de integración, una de las estrategias recomendadas es la de realizar pruebas incrementales. ¿En qué consiste esta estrategia?
    a. Probar un grupo de componentes dejándolos ya certificados, conforme avanza el desarrollo agregar los nuevos y así sucesivamente.
    b. Al igual que en las metodologías ágiles, implica ir, a la par que en el desarrollo, desarrollar – probar – implantar y avanzar con el siguiente grupo de entregables.
    c. Por cada nuevo componente que se agrega, es necesario probar todos.
8. Entre las estrategias de pruebas incrementales, tenemos la ascendente, descendente y combinada. ¿Cuál de ellas utiliza tres capas de componentes?
    a. Combinada.
    b. Ascendente.
    c. Descendente.
9. ¿En cuál de los siguientes aspectos se centran las pruebas del sistema?
    a. Integración de componentes.
    b. Seguridad, velocidad, exactitud.
    c. Las interfaces.

10. En relación con el equipo de pruebas del sistema ¿Cuál es la ventaja más significativa que se tiene si el equipo es externo?
    a. La independencia para generar los resultados.
    b. El desconocimiento del sistema.
    c. El equipo interno se puede dedicar de lleno al desarrollo.

Ir al solucionario

Resultados de aprendizaje 1 a 4:

- Conoce las principales áreas de conocimiento de la Ingeniería de *Software*.
- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.
- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de software.
- Identifica una metodología de desarrollo acorde a las características de un proyecto de software

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 15

Actividades finales del bimestre

Es momento de preparar la evaluación presencial, y al igual que en el primer bimestre le recomiendo que realice una revisión de las actividades de aprendizaje de las unidades 5 y 6, asegurándose de que comprende las temáticas tratadas, las actividades de final de unidad le permitirán afianzar su conocimiento.

Como estrategia de estudio, le recomiendo nuevamente elaborar mapas mentales, pero sobre todo llevar a cabo los ejercicios indicados, ya que estas unidades son más prácticas.

Resultados de aprendizaje 1 a 4:

- Conoce las principales áreas de conocimiento de la Ingeniería de *Software*.
- Conoce e Identifica las fases del ciclo de vida de desarrollo de sistemas.
- Identifica el ciclo de vida a utilizar en el desarrollo de un proyecto de software.
- Identifica una metodología de desarrollo acorde a las características de un proyecto de software

Contenidos, recursos y actividades de aprendizaje recomendadas

Semana 16

Actividades finales del bimestre

Para esta semana, se le recomienda repasar las unidades 7 y 8, tenga en cuenta que la unidad 8 tiene mucho contenido teórico, por lo que es recomendable hacerse un mapa mental del mismo.

La unidad 7 es quizá la más compleja desde el punto de vista técnico, por lo que es necesario que ponga más atención y repase los ejercicios de final de unidad planteados.

En caso de preguntas, no olvide el horario de tutorías o el uso del EVA para comunicarse con su tutor.

4. Solucionario

Autoevaluación 1

Pregunta	Respuesta	Retroalimentación
1	C	De acuerdo al artículo de Wirth, la complejidad del software se incrementó con el uso de sistemas de tiempo compartido y en ese entonces los programadores no tenían las herramientas o metodologías de desarrollo adecuadas para resolverlo, por ello la complejidad hizo que los proyectos se retrasen y acumulen muchos fallos que afectaron mucho a las empresas.
2	B	Corresponden a la segunda era, donde el software era considerado como producto y había explotado la crisis del software, por lo que se desarrollaron sistemas operativos multiusuario.
3	B	SOA significa Arquitectura Orientada a Servicios y de acuerdo al documento del OCW, las características son las planteadas en la alternativa B y la integración de soluciones con distintos proveedores, esto solo era posible mediante este tipo de arquitectura. La computación en la nube es posterior a SOA, y las otras alternativas no tienen que ver con SOA.
4	A	En efecto, el software no es solamente los programas, y desde el punto de vista de la Ingeniería de Software incluye los tres elementos mencionados, la alternativa d incluye personas y procesos que no corresponden al software y las demás están incompletas. Revisar OCW, Tema 1 diapositiva 9.
5	D	Todas las características, excepto la d, son aplicables al software, de hecho, el software es uno de los productos con mayor grado de incertidumbre durante su proceso desarrollo.
6	D	En esta alternativa hay dos errores, primero el cliente no siempre sabe lo que necesita ni conoce el proceso, por lo tanto, no puede indicar paso a paso lo que necesita y segundo un enfoque secuencial está demostrado que no es la mejor manera de abordar un proyecto de software.
7	C	La ingeniería de software se creó precisamente con el propósito de elevar la calidad de los productos de software, por lo tanto, todos los demás son efectos colaterales.
8	A	La principal razón para usar modelos en cualquier rama de la ciencia es incrementar la comprensión de sistemas complejos,

Pregunta	Respuesta	Retroalimentación
		aunque lo demás es consecuencia del modelado, este no siempre es visual, muchos modelos pueden ser matemáticos y no necesariamente derivan en documentación.
9	D	Aunque podría ser deseable siempre usar modelos, estos tienen más sentido en sistemas grandes y complejos, en las demás alternativas no habría un aporte significativo de los modelos.
10	D	El dominio de la aplicación corresponde a la comprensión del negocio, y esto se hace precisamente en el levantamiento de requerimientos y se consolida en el análisis.

Ir a la autoevaluación

Autoevaluación 2

Pregunta	Respuesta	Retroalimentación
1	a	Los usuarios del sistema son los que permiten al analista establecer los servicios, identificar las restricciones y plantear las metas del sistema a desarrollar, por lo tanto, la participación del usuario es fundamental, por tal motivo la etapa que requiere una participación directa del usuario en el modelo en cascada es en la etapa de análisis y definición de requerimientos.
2	a	En el modelo en cascada cada fase contempla el desarrollo de documentos debidamente autorizados que básicamente son los insumos de la siguiente fase, esto no quiere decir que, si en algún momento se necesita hacer algún alcance, se puede hacer siempre y cuando finalice una actividad.
3	b	En el desarrollo incremental la especificación, el desarrollo y la validación están entrelazadas, que permite realizar una serie de pasos hacia la solución y retrocede cuando se detectan errores. Cada incremento incorpora algunas funciones que necesita el cliente, siendo los primeros incrementos los que incluyen las funciones más importantes o las más urgentes.
4	c	La ingeniería de requisitos es una disciplina que aporta al proceso de desarrollo de software con la definición de los requerimientos, por tal motivo considera inicialmente realizar el estudio de factibilidad para analizar la conveniencia de realizar o no el sistema, luego centra sus esfuerzos en la obtención de información y el análisis mediante modelos de requisitos, seguidamente se realiza la especificación de los requerimientos y finalmente validarlos.
5	b	Los enfoques orientados a la reutilización se apoyan en componentes ya desarrollados que se pueden reutilizar y que pueden servir para diferentes aplicaciones. En este sentido, los servicios web, se pueden utilizar para desarrollos basados en componentes, ya que son una tecnología que utilizan un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
6	c	En la etapa de especificación del software, se determinan los requerimientos, por lo tanto, es necesario validarlos con la intervención del usuario, ya que las definiciones son a nivel de usuario.
7	a	El diseño arquitectónico, al ser la primera etapa en el proceso de diseño de software, por lo tanto, es la etapa donde se entiende cómo debe organizarse el sistema y cómo tiene que diseñarse la estructura global de ese sistema.

Pregunta	Respuesta	Retroalimentación
8	c	Cuando se ha establecido cómo funcionará cada componente, se procede con el diseño de la base de datos, dónde se diseñan las estructuras del sistema de datos y cómo se representarán en una base de datos en particular.
9	a	Cuando el desarrollador construye algún componente, procede a validar, mediante pruebas independientes, sin la integración de los restantes componentes. Estas pruebas realizadas por los desarrolladores se conocen como pruebas de desarrollo.
10	b	Los prototipos en la Ingeniería de requisitos son una versión inicial del sistema, de tal manera que el usuario puede tener una idea clara de lo que el sistema realizará, así como validar los requisitos.

Ir a la autoevaluación

Autoevaluación 3

Pregunta	Respuesta	Retroalimentación
1	C	Un Proyecto es un esfuerzo que tiene un propósito que resuelve una necesidad de negocio, las otras alternativas se refieren a actividades que pueden o no estar asociadas con un proyecto. Recuerde que los trabajos que tienen que ver con el desarrollo, modificación o mantenimiento de un producto de software, se constituyen en un proyecto.
2	B	El grupo de personas que se conforman para trabajar en un proyecto se denomina equipo del proyecto. Cada persona es un participante, los usuarios no trabajan en el proyecto, aunque sí aportan las necesidades y los interesados opinan o son afectados por el trabajo del proyecto, pero no forman parte del equipo de proyectos.
3	A	El alcance del proyecto es global y comprende tanto lo relacionado con el proyecto como lo relacionado con el producto, las demás alternativas son incorrectas. Revisar apartado 3.1
4	B	El papel de los interesados es esencial durante todo el proyecto, y pueden tener alta influencia e incluso pueden detener el desarrollo del proyecto. Las demás alternativas son falsas.
5	D	Un entregable es un producto de trabajo que se entrega a los clientes, en este caso los productos de trabajo a, b y c son para uso interno del equipo del proyecto.
6	A	En la etapa de definición, esta ocurre antes de la etapa de inicio y el arquitecto con el gestor del proyecto y otro usuario clave intentan pensar en lo que será el sistema y concebir una idea de lo que será la arquitectura del sistema.
7	B	Debido a que trabaja con el equipo de desarrollo y también con el lado del cliente para asegurarse que el producto hace lo que se espera.
8	C	En efecto, cuando se dibuja un cronograma, bien sea en un diagrama Pert o en un diagrama de Gantt, la ruta crítica se dibuja y resalta con color rojo.
9	A	El diagrama de Gantt es una matriz en la que se representa verticalmente las actividades y horizontalmente la escala de tiempo, las barras reflejan cuándo se desarrollará cada actividad en la escala de tiempo.
10	D	El diagrama Pert es un diagrama que forma un grafo en el cual los nodos son las actividades y las dependencias entre estas se

Pregunta	Respuesta	Retroalimentación
		visualizan por líneas que las unes, por tanto, estas dependencias se visualizan mejor en este tipo de diagramas, el diagrama de barras en el mismo que el de Gantt y el diagrama de actividades no se usa para representar cronogramas.

Ir a la autoevaluación

Autoevaluación 4

Pregunta	Respuesta	Retroalimentación
1	c	Para que los requisitos que se han establecido reflejen las necesidades reales del cliente, se debe contar con la información necesaria que sustente y justifique su definición, de no ser así, se tendrá que utilizar las técnicas necesarias para obtener la información adecuada.
2	b	Los requerimientos de sistema son definiciones más detalladas que permiten conocer con exactitud lo que se implementará. No así como los requerimientos de usuario o negocio que son enunciados más generales.
3	c	Los requerimientos no funcionales, no se relacionan directamente con los servicios del sistema, pero sí inciden y se derivan de las necesidades del cliente, por lo tanto, se definen en función del producto, de la organización y factores externos al sistema.
4	c	Las restricciones que considera los requerimientos no funcionales son de temporización, del proceso de desarrollo y aquellas impuestas por los estándares.
5	a	Cuando se hace referencia al software mediante especificaciones no funcionales como por ejemplo en cuanto a la rapidez, memoria requerida, tasa aceptable de fallas, etc., se refiere al producto.
6	a	Al hacer referencia a la cantidad de usuarios concurrentes, necesariamente se está incidiendo en la capacidad del software para operar bajo este requerimiento, por lo tanto, la respuesta correcta es el producto.
7	b	La herramienta de testing permitirá cumplir con una actividad del proceso de desarrollo de software, por lo tanto, se relaciona con la organización.
8	c	Cuando el software debe cumplir con una ley, como en este caso para los discapacitados, entonces son factores externos que determinan lo que se debe considerar en el sistema, por lo tanto, se refiere a requerimientos externos.
9	a	Para determinar los requerimientos es necesario interactuar con aquellas personas que realizan cada una de las actividades del proceso. Si bien puede haber participación de ciertas personas, ya sea para clasificar y priorizar los requerimientos, la mayor participación e incidencia radica en el descubrimiento de requerimientos.

Pregunta	Respuesta	Retroalimentación
10	b	Los prototipos permiten mostrar una versión lo más cercana a la versión final del software, por lo tanto, los usuarios podrán experimentar con la funcionalidad y verificar que cumple con lo que necesitan.

Ir a la autoevaluación

Autoevaluación 5

Pregunta	Respuesta	Retroalimentación
1	b	Los casos de uso representan las funciones que cumplirá el sistema desde el punto de vista del usuario, a esta parte se le conoce como modelo funcional, por tanto, la respuesta correcta es la B. El modelo de objetos representa las entidades identificadas en el contexto del sistema y el modelo dinámico representa la interacción entre objetos, por lo tanto, ninguna de las dos alternativas corresponde al modelo funcional.
2	c	El modelo dinámico está conformado por dos tipos de diagramas, los diagramas de secuencia y los diagramas de estados, los primeros representan las interacciones entre los diferentes objetos que participan en un escenario del sistema, y los segundos representan el estado interno de un objeto, cuyos cambios se dan por efecto de eventos que alteran su comportamiento, por lo tanto, la respuesta correcta es C, diagramas de estados. Los diagramas de clases no forman parte del modelo dinámico.
3	a	Los objetos del análisis se clasifican en categorías, los objetos borde (boundary), tienen la función de proveer mecanismos de interacción del sistema con usuarios u otros sistemas, por lo tanto, se usan para representar elementos de interacción como pantallas o archivos de intercambio de datos, los objetos establecidos como entidad, corresponden a los objetos cuya información y comportamiento reflejan el proceso de negocio, por tanto, se excluyen como respuestas válidas a la b y la c, los objetos de control por su parte describen el comportamiento del sistema integrando a los otros dos, a este proceso se lo conoce como realización de casos de uso, por tanto, la respuesta correcta es la A.
4	c	De acuerdo a las características estudiadas para cada tipo de objetos, en la opción A, ícono no puede ser un objeto entidad, lo cual la descarta, la alternativa B por su lado coloca erróneamente al objeto puntero como objeto de control, lo cual es incorrecto, por lo tanto, la alternativa correcta es la C.
5	a, d, f	Las entidades del software se caracterizan porque no tienen una contraparte en el mundo físico, por ello las alternativas A, D y F cumplen con esta condición, es decir en el mundo físico no existen elementos como, LocalizadorGPS, BasedeDatosZonaHoraria, IdentidadUsuario.
6	a	A pesar de obtener el mismo resultado en términos de implementación, el origen de estas clases difiere. La generalización se refiere al proceso por el cual se identifica características comunes a varios objetos y se crea una superclase con el propósito de reutilizarlas optimizando el

Pregunta	Respuesta	Retroalimentación
		modelo, en este tipo de relación por lo general la superclase se constituye en una clase abstracta. En el caso de la especialización, se parte de una clase concreta ya definida en la cual identificamos otros objetos que requieren de algún comportamiento particular, y para ello se define una nueva clase que agrega los métodos y atributos necesarios para conseguir ese nuevo comportamiento, el cual puede ser una modificación del original o nuevas operaciones.
7	b	Analizando el texto de la pregunta “está conformada por”, hace referencia a una relación “consiste de” que corresponde a una agregación, luego la agregación puede ser de dos tipos, la agregación por composición, la cual se refiere a una relación estructural en la cual las partes dependen del todo y la agregación compartida, en la cual el todo puede existir independientemente de las partes, la cual corresponde al caso propuesto.
8	a	En efecto, los objetos borde, son los únicos que pueden interactuar con los actores, que en este caso corresponde al actor cliente, los objetos de control no tienen la posibilidad de comunicarse con los actores, sino a través de un objeto borde.
9	c	El criterio de estabilidad se refiere a que se ha cumplido con los requerimientos establecidos y que los cambios se han minimizado. Es muy difícil llegar a una situación en la que ya no se den más cambios, sin embargo, si esto se minimizan, se puede dar por estabilizada la etapa y, por tanto, concluida.
10	b	Cada participante de un equipo de desarrollo tiene sus roles bien definidos, los analistas levantan y analizan los requerimientos, los programadores implementan, quien propone la solución vinculando los casos de uso con el modelo de objetos es el arquitecto.

Ir a la autoevaluación

Autoevaluación 6

Pregunta	Respuesta	Retroalimentación
1	b	En el capítulo 6, el texto básico expone que la arquitectura en pequeño se preocupa de los programas individuales, en tanto que la arquitectura en grande se encarga de modelar componentes de grandes sistemas. El Modelo Vista Controlador (MVC) es un estilo cuyo fin es separar los componentes de datos, de la lógica de negocio y de la visualización, por lo tanto, este aplica a grandes aplicaciones empresariales, que es lo que indica la alternativa B.
2	c	Los requerimientos funcionales corresponden a las operaciones que debe realizar el usuario y corresponden a la lógica de negocio. Estos se implementan en componentes individuales, ya que a nivel de arquitectura se representan los requerimientos no funcionales.
3	a	El desarrollar la arquitectura del sistema tiene ventajas evidentes en diferentes etapas y aspectos, en capítulo 6 del texto básico se establecen tres ventajas de este enfoque, una de las cuales se manifiesta la comunicación con los participantes, y establece que esto se da porque se trata de una presentación a alto nivel del sistema, que puede usarse para discutir sobre las decisiones del sistema.
4	a	El modelo de análisis es un paso intermedio, que permite tomar los requerimientos identificados en la fase de levantamiento y los transforma en conceptos compatibles con aplicaciones de software, sin embargo, no representa la estructura del sistema, esto lo hace el diseño.
5	c	La disponibilidad es un atributo de calidad que establece que el sistema estará disponible la mayor parte del tiempo, y en una forma de garantizar ante posibles fallos es la redundancia, establecida en la alternativa seleccionada.
6	a	Cada atributo de calidad se enfoca en diferentes características del sistema, muchas de las cuales pueden entrar en conflicto con otras, por ejemplo, si se quiere tiempo de facilidad de uso, se puede contraponer con la seguridad, Ej. En las aplicaciones bancarias, por más usables que sea, la seguridad obliga a que se coloquen controles como contraseñas, imágenes, preguntas secretas, etc. que hacen que la facilidad de uso se vea afectada.
7	c	El modelo propuesto por Kruchten comprende 5 vistas, una vista lógica, una vista de procesos, la vista física, la vista de desarrollo, y la vista de escenarios de casos de uso. De ellos la vista lógica representa el modelo de objetos y clases.

Pregunta	Respuesta	Retroalimentación
8	b	Existen diferentes tipos de lenguajes para representar los modelos del sistema, el lenguaje UML se usa para representar diferentes modelos del sistema, entre ellos la arquitectura, pero no logra un nivel de detalle suficiente; el lenguaje C es un lenguaje de programación y los ADL son lenguajes de descripción de la arquitectura y están destinados específicamente para ese propósito.
9	c	La metodología de desarrollo se enfoca en la forma de organizar las actividades de desarrollo, el detalle de la arquitectura es un elemento de construcción del producto que puede hacerse bien sean mediante un enfoque ágil o un enfoque tradicional. Esto depende del nivel de complejidad del producto que se esté desarrollando.
10	b	Las recomendaciones generales sobre cohesión y acoplamiento indican que se debe minimizar el acoplamiento y maximizar la cohesión, sin embargo, es preciso considerar que cualquier decisión en esa línea afecta a la otra, es decir, para incrementar la cohesión, es preciso reducir la cohesión, por tanto, lo más recomendable es usar la heurística 7 ± 2, es decir entre 5 y 9 niveles de descomposición en subsistemas.

Ir a la autoevaluación

Autoevaluación 7

Pregunta	Respuesta	Retroalimentación
1	b	Los objetivos del diseño se centran específicamente en resolver los requerimientos no funcionales, que darán soporte a los requerimientos funcionales, los cuales se deben implementar en los componentes.
2	a	Cuando hablamos de contexto, siempre nos referimos a una vista global del sistema, y este, por tanto, no puede mostrar la parte interna del sistema, sino sobre todo los sistemas externos, las restricciones no tienen cabida en esta apreciación.
3	c	Uno de los elementos más importantes del diseño es descomponer el sistema en subsistemas que cumplan características mínimas como la cohesión, acoplamiento y puedan organizarse en alguno de los estilos arquitectónicos estudiados.
4	c	Las respuestas a eventos están relacionadas con el comportamiento de los objetos, con la cual la única alternativa es la de las máquinas de estados.
5	b	Cada uno de los diagramas de UML tiene un propósito: los diagramas de componentes muestran los componentes lógicos que conforman el sistema, los diagramas de actividades muestran las secuencias de operaciones que deben realizarse y los diagramas de implantación, precisamente muestran los dispositivos con los componentes que correrán en cada uno de ellos.
6	a	El almacenamiento persistente se hace sobre aquellos elementos que necesitamos almacenar y actualizar en el tiempo, los denominados objetos borde y de control solo existen en tiempo de corrida, por tanto, los objetos que pueden utilizarse para este tipo de almacenamiento son los objetos entidad.
7	c	Las bases de datos orientadas a objetos se especializan en almacenar los objetos como entidades completas, sin embargo, estas estructuras son complejas y las operaciones que deben realizarse se ven afectadas en el rendimiento, y uno de los motivos es porque no hay un modelo matemático que le sirva de base como sucede con las bases de datos relacionales.
8	c	La encriptación consiste en utilizar un algoritmo que oculta la información reemplazándola por un contenido ilegible por parte de un tercero, lo cual significa que si alguien logra robar, no la podrá entender a no ser que la desencripte.
9	b	

Pregunta	Respuesta	Retroalimentación
		Por definición, un patrón recoge un conjunto de buenas prácticas aplicado en diferentes casos para resolver problemas particulares, en este caso se trata de problemas de diseño.
10	c	Las herramientas de administración de la configuración permiten mantener diferentes versiones de componentes que pueden ser utilizados por varios participantes guardando la consistencia y preservando el control de cambios, de este modo evitan conflictos en el uso de dichos componentes.

Ir a la autoevaluación

Autoevaluación 8

Pregunta	Respuesta	Retroalimentación
1	c	En la introducción al capítulo 8 del texto básico, se establece los conceptos de verificación y validación, los cuales a pesar de que suelen confundirse tienen objetivos diferentes. La verificación se enfoca en comprobar que el software cumpla con sus especificaciones (funcionalidades y requerimientos no funcionales), por su parte, la validación se enfoca en asegurar que se cumpla con las expectativas del cliente, es decir, que el software le sea útil. (pág. 207 texto básico).
2	a	Las inspecciones no sustituyen a las pruebas del software, debido a que no son suficientes al momento de detectar defectos surgidos por interacciones inesperadas. (pág. 209, texto básico).
3	b	La presencia de errores en las primeras versiones del producto suelen darse como un comportamiento normal. Sin embargo, si este número de detecciones incrementa de manera considerable, la única conclusión a la que se puede llegar es que podrían presentarse errores en componentes futuros; las otras alternativas no hay forma de demostrar que sean ciertas.
4	c	De acuerdo a la definición de fallo dada en los conceptos de inicio del presente capítulo, la C sería la alternativa correcta.
5	a	Las pruebas de integración y de sistema operan poniendo a funcionar varias partes del sistema, por tanto, la única alternativa válida sería que operan a nivel de componentes.
6	c	Las pruebas de componentes se diferencian de las pruebas de unidad específicamente en que intentan determinar la validez y funcionamiento correcto de las interfaces con otros componentes.
7	c	Como se ilustra en la figura 36 de la guía didáctica, las pruebas incrementales implican probar todo lo anterior más el componente que se agrega.
8	a	La estrategia combinada utiliza la estrategia ascendente y descendente y el sistema se presenta como un sándwich con 3 capas. La capa central es la que se desea probar, mediante enfoque ascendente prueba la capa central, asumiendo que la inferior ya ha sido implementada, y se desciende a la capa de prueba desde la capa superior.
9	b	Una vez realizadas las pruebas de componentes donde se ha determinado la validez de los componentes y las interfaces, el siguiente nivel son las pruebas del sistema, cuyo enfoque es

Pregunta	Respuesta	Retroalimentación
		precisamente aspectos como la velocidad, seguridad, exactitud, entre otros.
10	a	La ventaja más importante es precisamente la independencia de las pruebas, que es una característica deseable en cualquier equipo de pruebas o de auditoría. Las otras dos alternativas que son ciertas, no representan una ventaja.

Ir a la autoevaluación

5. Referencias Bibliográficas

Ahmed, A. (2018). Software Engineering in the Agile World (1st ed.). Independently published.

Ahmed, A., & Bhanu, P. (2016). Foundations of Software Engineering. In National Conference Publication - Institution of Engineers, Australia. Boca Raton: Taylor & Francis.

Bruegge, B., & Dutoit, A. H. (2010). Object-Oriented Software Engineering Third Edition (3rd ed.). EE. UU.: Prentice Hall.

Gottesdiener, E. (2005). The software requirements memory jogger (1st ed.). Salem: GOAL/QPC.

Institute of Electrical and Electronics Engineers. (1990). IEEE Standard Glossary of Software Engineering Terminology (Vol. 121990). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342%2520

Kruchten, P. (2000). The Rational Unified Process: An introduction (2nd ed.). Massachusetts: Addison Wesey.

Kung, D. C. (2014). Object-oriented Software Engineering: An Agile Unified Methodology. New York: McGrawHill.

Microsoft Corporation. (2009). Microsoft Architecture Application Guide Patterns & Practices (2nd ed.).

Project Management Institute, P. (2017). Project Management Body Of Knowledge and Agile Practice Guide (1st ed.; P. M. Institute, ed.). Pennsylvania.

Ruparelia, N. B. (2010). Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, 35(3), 8. https://doi.org/10.1145/1764810.1764814

Sánchez, S., Sicilia, M. Á. & Rodríguez, D. (2012). Ingeniería del Software. Un enfoque desde la guía del SWEBOK (1st ed.; A, ed.). Bogotá: Alfa Omega.

Schwaber, K., & Sutherland, J. (2020). La Guía de Scrum. La Guía Definitiva de Scrum: Las Reglas del Juego. 22. Retrieved from https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf

Scrum.org. (2020). What-is-scrum @ www.scrum.org. Retrieved from https://www.scrum.org/resources/what-is-scrum

Society, I. C. (2014). Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK Guide V3.0) (P. Bourque & F. Richard, Eds.). Retrieved from https://www.computer.org/education/bodies-of-knowledge/software-engineering

Wirth, N. (2008). A Brief History of Software Engineering. IEEE Annals of the History of Computing, 30(3), 32–39. https://doi.org/10.1109/MAHC.2008.33