**Datos de identificación del programa de formación**

| PROGRAMA DE FORMACIÓN | Desarrollo de aplicaciones web full stack |
| --- | --- |

| COMPETENCIA | 220501092 - Definición de los requerimientos del *software.* | RESULTADOS DE APRENDIZAJE | 220501092-02 - Construir el informe de requisitos de la solución conforme a la identificación de las necesidades del negocio. |
| --- | --- | --- | --- |

| NÚMERO DEL COMPONENTE FORMATIVO | 2 |
| --- | --- |
| NOMBRE DEL COMPONENTE FORMATIVO | Construcción del informe de requisitos |
| BREVE DESCRIPCIÓN | Esta fase implica determinar los estándares de especificación de requisitos, con el fin de encontrar las mejores técnicas de análisis para una buena realización de resultados en los estudios, de esta manera, mejorar el desarrollo del informe según las especificaciones de requisito de *software*, y llevar un control de todas las versiones de los informes, de forma organizada y controlada. |
| PALABRAS CLAVE | Requisitos, planeación, informe |

| ÁREA OCUPACIONAL | 6 - VENTAS Y SERVICIOS |
| --- | --- |
| IDIOMA | Español |

# **TABLA DE CONTENIDOS**

**1. Metodologías de desarrollo de *software***

**2. Lenguaje unificado de modelado**

**3. Requerimientos de *software***

**4. Técnicas de documentación de requerimientos en proyectos ERS**

**INTRODUCCIÓN**

| Cuadro de texto |
| --- |
| Apreciado aprendiz, bienvenido a este componente formativo donde tendremos la oportunidad de explorar, a través de conceptos, las principales metodologías de desarrollo ágil y sus tipos; así mismo, el buen uso de la documentación en los proyectos ágiles, los tipos de lenguaje modelado, sus técnicas de análisis, características, ventajas y desventajas. Y, por último, tener en cuenta los requisitos de *software* necesarios para un buen desarrollo.  En el siguiente video conocerá, de forma general, la temática que se estudiará a lo largo del componente formativo. |

**GUION DE VIDEO INTRODUCTORIO**

| **Tipo de recurso** | Video motion | | | |
| --- | --- | --- | --- | --- |
| **NOTA** |  | | | |
| **Título** | Introducción | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración** | **Texto** |
| **1** | Metodología de desarrollo ágil concepto de negocio Ciclo de vida ágil para el diagrama de desarrollo de software Eficacia del trabajo en equipo para sprint de proyectos Ciclo de reglas de programación adaptativa y estrategia de administración de procesos |  | En este componente formativo podrá aprender, conocer y diferenciar los esquemas principales de las metodologías de desarrollo ágil; cuáles son las más utilizadas a nivel de proyectos y cuál de estas se aplica mejor, de acuerdo con las necesidades del cliente. | Metodologías de desarrollo ágil |
| **2** | Descripción del requisito de software descripción ilustración abstracta del vector de concepto. Descripción del sistema de software, herramienta ágil, análisis de negocios, especificaciones de desarrollo de proyectos, metáfora abstracta de documentos. |  | Así mismo, aporta el conocimiento sobre las técnicas de documentación de requisitos en proyectos con metodología tradicional y metodologías ágiles, además de sus principales características. Esto se debe a que es necesario conocer primero las diferencias entre los tipos de metodologías y todo lo que se abarca en ellas. | Técnicas de documentación |
| **3** | CMS, concepto de arquitectura de software. Administración de sistemas de gestión de contenidos, estructura de red, optimización de procesos en línea en el negocio. Ilustración vectorial plana aislada en fondo blanco  UML - Lenguaje de modelado unificado. concepto de negocios de siglas. concepto de ilustración vectorial con palabras clave e iconos. ilustración con iconos para banner web, volante, página de inicio |  | Este componente permite obtener un soporte robusto en el desarrollo y debe identificarse desde un inicio; esto se lleva a cabo para garantizar una configuración alrededor del proyecto organizacional, lo que permite un buen funcionamiento de esta misma estructura. Estos componentes deben ser comprendidos en su totalidad porque son una parte importante de su desempeño. Los modelados de lenguaje, conceptos y tipos son necesarios para aprender a modelar un diseño de *software* orientado a objetos con el fin de dar mayor profesionalismo al desarrollo del *software* trabajado; lo cual implica tener una mejor oportunidad en el mercado laboral por dominar estos conocimientos. | Buen funcionamiento de la estructura  Modelado de lenguaje |
| **4** | Análisis de requisitos en el desarrollo de negocios o sistemas, creación de requisitos de software y especificación que describe la tarea de usuario en el documento con equipo |  | Debido a la gran variedad de plataformas y sistemas operativos podemos encontrarnos con que el *software* no es uniforme, pero esto no significa que no tenga las mismas características. Los requisitos de un *software* pueden ser los mismos para todos, pero para poder cumplir con los requisitos, primero debemos saber cuáles son para poder trabajar con ellos como objetivos y así cumplirlos para que el *software* tenga cierto nivel de calidad, para que no falle en el mismo momento de ser utilizado por el usuario final. Esto es muy importante y necesario, ya que un *software* que ha fallado y se ha derrumbado por sus propios defectos y problemas estructurales, no es algo de lo que los desarrolladores puedan presumir, sino algo en lo que tienen que trabajar y componer. | Requisitos de un *software* |
| **5** | Especificaciones de desarrollo de proyectos conjunto de ilustraciones vectoriales de concepto abstracto. Análisis de negocio, descripción de requisitos de software, documento de visión y ámbito, análisis SWOT, metáfora abstracta de caso de usuario. |  | A medida que los años van pasando y las tecnologías van evolucionando, el *hardware* se hace cada vez más robusto y, por ende, el *software* se tiene que ir actualizando a la medida de ese crecimiento. Los requisitos han evolucionado de la mano de estos, pero aún así deben cumplir con ciertas características principales y ciertos estándares de desarrollo, con el fin de brindar una calidad de *software* mínima al usuario final.  Para llegar a esto es necesario conocer el detalle de sus requisitos, sus clasificaciones y las ventajas que esto conlleva en el desarrollo del *software,* para que el desarrollador pueda trabajar sobre lo planteado y así cumplir las necesidades de forma satisfactoria. | Evolución de los requisitos  Calidad de *software* |
| **6** | Metodología del ciclo de vida del desarrollo de software y del sistema de modelo V. Proceso de gestión de proyectos desde el diseño, la implementación hasta la fase de integración y prueba. |  | Al mismo tiempo, es importante tener en cuenta todas las mejores prácticas y normas de calidad del código, para conseguir los mejores resultados. En este componente formativo se analizan los diferentes aspectos de los requisitos de *software* y lo importante que es tener un enfoque estandarizado para el desarrollo de s*oftware*.  ¡Bienvenido al estudio y comprensión de este componente y suerte en su recorrido! | Requisitos de *software* |
| **Nombre del archivo** | 228125\_v1 | | |  |

**DESARROLLO DE CONTENIDO**

**1. Metodologías de desarrollo de *software***

| Cuadro de texto |
| --- |
| Qué mejor manera de empezar el componente formativo que con las metodologías de desarrollo de *software* y sus derivados, y así validar cuál metodología se puede abarcar según sus necesidades.  Lo invitamos a ver el siguiente video para conocer qué es una metodología de desarrollo de *software*. |

| **Tipo de recurso** | Video motion | | | |
| --- | --- | --- | --- | --- |
| **NOTA** |  | | | |
| **Título** | Metodología de desarrollo de *software* | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración** | **Texto** |
| **1** | Metodologías del Desarrollo de Software |  | En los espacios donde se trabaja un proyecto, los equipos, para desplegar su trabajo, utilizan metodologías de desarrollo de *software*.  Se define metodología de desarrollo de *software* como una agrupación de sistemas y métodos ordenados que se usan para diseñar resultados de *software* informático. El objetivo de las distintas metodologías es el de intentar organizar los equipos de trabajo para que estos desarrollen las funciones de un programa, de la mejor manera posible. | Metodologías de desarrollo de *software*  Organizar los equipos de trabajo |
| **2** | Cómo organizar tu día de trabajo en 6 pasos. – IMATION |  | Estas metodologías han ido cambiando, y han pasado de ser una formalidad de estructura, a ser un apoyo importante cuando se va a desarrollar *software* productiva y eficientemente. Permiten organizar el trabajo de manera ordenada porque cuentan con factores como costos, planeación, lenguaje, equipo de trabajo; para poder desarrollar los productos o solucionar conflictos para los clientes o para el mercado con el que se desenvuelven. | Organizar el trabajo  Costos  Planeación  Lenguaje  Equipo de trabajo |
| **3** | Metodologías ágiles para el desarrollo de software – 1 de 3 – Hipervinculos |  | Es importante contar con una metodología clara para que al desarrollar un *software* no se presenten problemas, errores y un mal resultado. Trabajar basándose en una buena metodología reduce las dificultades, permite organizar las tareas y agilizar el procedimiento para obtener un buen resultado final del proyecto que se está llevando a cabo.  Cuando se habla de metodologías de desarrollo de *software*, hay que conocer sobre ellas y saber que se presentan dos grandes grupos. | Reduce las dificultades  Organizar tareas  Agilizar el procedimiento |
| **4** | Metodologías tradicionales y Metodologías ágiles - Diego Calvo |  | El primero es el grupo de las metodologías de desarrollo de *software* tradicionales.  Estas se caracterizan porque definen los requisitos, al iniciar los proyectos, de una manera estricta y rígida. Hay poca flexibilidad en los ciclos de desarrollo y no se permiten cambios.  Su estructura para trabajar es lineal, o sea que las etapas van una tras otra y se debe terminar la anterior antes de continuar con la siguiente. No se puede retroceder en una etapa y no se adapta a los cambios, lo que no es acorde con el cambio permanente en las tecnologías. | Metodologías de desarrollo de *software* tradicionales  Estricta y rígida  Estructura lineal |
| **5** | Ciclo de Vida Modelo en Cascada - YouTube |  | Ejemplos de estas metodologías son la metodología cascada, porque las etapas se organizan de arriba a abajo; y prototipado, porque se fundamentan en construir un prototipo de *software* rápidamente y los usuarios puedan probarlo y aportar *feedback.* | Cascada  Prototipado |
| **6** | Introducción a las metodologías ágiles |  | El segundo, es el grupo de las metodologías de desarrollo de *software* ágiles.  Su estructura de trabajo las ubica como las más utilizadas por su flexibilidad y agilidad. Se caracterizan por ser más productivas y eficientes, permiten que el *software* se adapte a las necesidades y esto facilita nuevas aplicaciones con mejor funcionalidad. | Metodologías de desarrollo de *software* ágiles  Flexibilidad  Agilidad  Productivas  Eficientes |
| **7** | Metodologías Ágiles | Cátedra Viewnext USAL |  | Las metodologías ágiles se fundamentan en la metodología del incremento, donde en cada ciclo de desarrollo se adicionan nuevas funciones. Como los ciclos son más cortos y rápidos, se van adicionando pequeñas operatividades en vez de grandes cambios. Desde esta forma, se permiten equipos de trabajo autosuficientes, independientes, el cliente puede aportar nuevas necesidades porque tiene acceso a la evolución del proyecto.  Trabajar con una metodología de desarrollo de *software* permite reducir el nivel de dificultad, organizar las tareas, agilizar el proceso y mejorar el resultado final de las aplicaciones a desarrollar. | Metodología del incremento  Ciclos cortos y rápidos  Equipos de trabajo autosuficientes e independientes  Cliente puede aportar |
| **Nombre del archivo** | 228125\_v2 | | |  |

**Tipos de metodología de desarrollo de *software* tradicionales**

| Cuadro de texto |
| --- |
| Son aquellas que **no se ajustan nada bien a los cambios** y su comportamiento resalta de forma muy superficial. La siguiente imagen presenta el flujo de una metodología de desarrollo de *software* tradicional. |

**Figura 1**

*Flujo de una metodología de desarrollo de software tradicional*

****

| **Tipo de recurso** | Pestañas o tabs horizontales | |
| --- | --- | --- |
| **Introducción** | Conozcamos algunas de las principales metodologías tradicionales: | |
| ***Waterfall*  (cascada)** | En esta metodología se despliegan las diferentes funciones en etapas separadas y obedeciendo un estricto orden. Antes de cada fase se debe verificar el producto para ver si está listo para pasar a la siguiente etapa. Las especificaciones y requisitos iniciales no están propensos a cambiarse, por lo que no se pueden ver los avances hasta que el proyecto ya esté en gran parte ejecutado. | Plantilla de diseño plano de página Web para el modelo de cascada. Metodología del ciclo de vida de la página de inicio del negocio sobre cómo aumentar las ventas. Concepto moderno de ilustración vectorial para el desarrollo de sitios web y sitios web móviles  Imagen: 228125\_i2 |
| **Prototipado** | Se basa en la ejecución de un prototipo de *software* que se desarrolla rápidamente para que los usuarios puedan aportar una retroalimentación. Así, se puede corregir lo que está mal e introducir otros requerimientos que sean necesarios. Es un modelo que se basa en el procedimiento de prueba y error para entender las especificidades del producto. | Desarrollo rápido de aplicaciones, icono vector  Imagen: 228125\_i3 |
| **Espiral** | Es la unión de los dos modelos anteriores, la diferencia es que esta, añade el concepto de análisis de riesgo. Se separa en cuatro fases: planificación, análisis de riesgo, desarrollo de prototipo y evaluación del cliente. El nombre de este tipo de metodología se da debido a su funcionamiento, porque se van ejecutando las etapas en forma de espiral, cuanto más cerca del centro significa que más avanzado está el proyecto. | Modelo espiral: historia, características, etapas, ejemplo  Imagen: 228125\_i4 |
| **Incremental** | En esta metodología se va generando el producto final de manera progresiva. En cada fase incremental se añade una nueva funcionalidad, lo que permite observar los resultados de una forma más rápida con respecto al modelo en cascada. El *software* se puede comenzar a utilizar incluso antes que se finalice totalmente y, en general, es mucho más flexible que las demás metodologías. | Modelos Evolutivos Del Software: Modelo Incremental  Imagen: 228125\_i5 |
| **Diseño rápido de aplicaciones (RAD)** | Esta metodología posibilita desarrollar *software* de alta calidad en un periodo de tiempo corto. El desarrollo es más flexible pero los costos son mucho más altos, aunque demanda una mayor intervención de los usuarios. Así mismo, el código puede incluir más errores y sus funciones pueden reducirse debido al poco tiempo del que se dispone para desarrollarlas. El objetivo principal es reproducir el menor número posible de veces para alcanzar una aplicación funcional, completa y de forma rápida. | Desarrollo rápido de aplicaciones metodología de software RAD, esquema detallado de vector de procesos de marco. Planificación de requisitos, diseño de usuario, prototipo, prueba y ajuste de bucle, construcción y corte.  Imagen: 228125\_i6 |

**Tipos de metodología de desarrollo de *software* ágiles**

| Cuadro de texto |
| --- |
| Ahora, conozcamos este tipo de metodologías, las cuales se basan en la metodología incremental, en la que en cada ciclo de desarrollo se va construyendo y puliendo el producto final, a la vez que el cliente puede ir conduciendo nuevos requerimientos debido a que puede comprobar cómo avanza el proyecto en tiempo real. La siguiente imagen presenta el flujo de una metodología de desarrollo de *software* ágil. |

**Figura 2**

*Flujo de una metodología de desarrollo de software ágil*

****

| **Tipo de recurso** | Pestañas o tabs horizontales | |
| --- | --- | --- |
| **Introducción** | Las principales metodologías ágiles son: | |
| **Kanban** | Esta metodología consiste en separar las actividades en fragmentos mínimos y organizarlas en un tablero de trabajo, separándolas en tareas pendientes, en curso y finalizadas. De esta manera, se genera un flujo de trabajo muy visual basado en tareas primordiales y generando un valor agregado al producto. | Vector illustration of people plan their schedule, priority task and checking an agenda. Woman and man working on big creen. An idea of Kanban board, time management  Imagen: 228125\_i8 |
| **Scrum** | Es también una metodología incremental que divide las tareas y requisitos de forma similar a Kanban. Se reproduce sobre bloques de tiempos fijos y cortos (entre dos y cuatro semanas) para alcanzar un resultado completo en cada iteración. Las fases son: planificación de la iteración (*planning sprint*), llevar a cabo (*sprint*), reunión diaria (*daily meeting*) y demostración de los avances (*sprint review*). A todas estas etapas se le denomina también *sprint*. | Infografía del proceso de escrutinio. Metodología de desarrollo ágil, administración de sprints y trabajo en tiempo real. Pictograma de distribución, desarrollo de tecnología premium metodologías de desarrollo ilustración vectorial  Imagen: 228125\_i9 |
| **Lean** | Está configurado para que grupos pequeños de desarrollo altamente capacitados puedan elaborar cualquier tarea en poco tiempo. Los activos más importantes son su compromiso y las personas del equipo, posponiendo así a un segundo plano los costes y el tiempo. | Agile Safe Lean  Imagen: 228125\_i10 |
| **Programación extrema (XP)** | Su principal objetivo es generar un buen ambiente de trabajo con el grupo y que haya una retroalimentación constante del cliente. El trabajo se basa en 12 fases: diseño fácil, pruebas, codificación con estándares y refactorización, propiedad colectiva del código, programación en par, integridad con el cliente, integración continua, entregas semanales, atención al cliente, planificación y entregas frecuentes. | Imagen: 228125\_i11 |

**2. Lenguaje unificado de modelado**

| Cuadro de texto |
| --- |
| Continuemos con los lenguajes unificados de modelado (UML), los cuales son una serie de normas y estándares que dicen cómo se debe representar algo de la vida real a un sistema de cómputo, sea a través de diagramas o expresiones computacionales que representan una ecuación matemática. Normalmente son usados por grupos grandes de desarrollo de *software* con el objetivo de planificar y documentar cómo se construyen los programas informáticos complejos. Los UML definen normas para construir muchos tipos de esquemas, no esquemas de un solo tipo. |

| **Tipo de recurso** | Carrusel de tarjetas | |
| --- | --- | --- |
| **Introducción** | Las ventajas del uso de UML se pueden definir de la siguiente manera: | |
| Ayuda a analizar el cómo se implementa. | | class diagram uml unified modeling language  **Imagen:** 228125\_i12 |
| Se facilita la comunicación entre el grupo al existir un lenguaje en común. | | class diagram uml unified modeling language  **Imagen:** 228125\_i12 |
| Se dispone de documentación que trasciende al proyecto. | | class diagram uml unified modeling language  **Imagen:** 228125\_i12 |
| Los modelos ayudan a visualizar cómo es o cómo se quiere que sea el sistema. | | class diagram uml unified modeling language  **Imagen:** 228125\_i12 |
| Especifican el comportamiento y la estructura del sistema. | | class diagram uml unified modeling language  **Imagen:** 228125\_i12 |
| Guían la construcción del sistema. | | class diagram uml unified modeling language  **Imagen:** 228125\_i12 |

| Cuadro de texto |
| --- |
| En los UML se pueden definir dos tipos principales de diagramas y estos a su vez se subdividen, conozcámoslos. |

**Diagramas estructurales UML**

| Cuadro de texto |
| --- |
| Hablemos primero de los diagramas estructurales UML, los cuales representan la configuración general de un sistema: sus atributos, relaciones y objetos entre las distintas entidades. Es decir, se preocupa más por lo que es el sistema y no por cómo se comporta. |

| **Tipo de recurso** | Slider Presentación | |
| --- | --- | --- |
| **Introducción** | Entre ellos podemos encontrar: | |
| **Diagrama de clases:** estos tipos de diagramas UML se utilizan para proporcionar la representación estática del programa. Una clase tiene tres elementos fundamentales: su comportamiento, su nombre y sus atributos. | | **Imagen:** 228125\_i13 |
| **Diagrama de componentes:** se encarga de desglosar todo el sistema en pequeños componentes, de tal manera que se pueda ver cómo dependen unos de los otros. Este tipo de diagramas constan únicamente de dos partes: las dependencias y los componentes. | | **Imagen:** 228125\_i14 |
| **Diagrama de despliegue:** está dividido en dos partes principales: los artefactos y los nodos. Los artefactos son los encargados de representar a los clientes o los esquemas, y por otro lado, los nodos son los que representan el servidor de la base de datos. | | **Imagen:** 228125\_i15 |
| **Diagrama de objetos:** presenta múltiples objetos en cada enlace. Cada entidad consta de su grupo de características y un nombre definido de objetos. | | **Imagen:** 228125\_i16 |
| **Diagrama de paquetes:** provee una manifestación de un sistema con sus respectivos módulos y los subsistemas que este conlleva de más alto nivel. | | **Imagen:** 228125\_i17 |
| **Diagrama de perfil:** se utiliza para mostrar la meta configurada del sistema. Además, puede generar conexiones entre los elementos de un perfil y diferentes perfiles. | | **Imagen:** 228125\_i18 |
| **Diagrama de la estructura compuesta**: se usan para reproducir la configuración interna de una clase. | | **Imagen:** 228125\_i19 |

**Diagramas UML de comportamiento**

| Cuadro de texto |
| --- |
| Ahora estudiemos los diagramas UML de comportamiento, los cuales determinan la funcionalidad general de cada unidad y el comportamiento del sistema. Además, algunos de estos diagramas manifiestan también el flujo de información y control en el sistema. |

| **Tipo de recurso** | Slider Presentación | |
| --- | --- | --- |
| **Introducción** | Entre ellos podemos encontrar: | |
| **Diagrama de máquina de estado:** define los tipos de estados del sistema. Además, muestra las acciones, sus tipos y cómo resolvería una clase a ellas. | | **Imagen:** 228125\_i20 |
| **Diagrama de actividad:** manifiesta el flujo general del sistema en relación con varias actividades. Recibe una instancia del sistema y observa cómo reacciona. | | **Imagen:** 228125\_i21 |
| **Diagrama de casos de uso:** demuestra cómo el usuario final se comunicaría con el sistema. Además, genera múltiples actores principales, como por ejemplo, los usuarios y su comunicación con elementos puntuales del sistema. | | **Imagen:** 228125\_i22 |
| **Diagrama secuencial:** especifica el flujo a través de un proceso entre múltiples componentes de manera consecutiva, presentando variables elementos y especificando el flujo que pasa de un componente a otro. | | **Imagen:** 228125\_i23 |
| **Diagrama de comunicación:** manifiesta de forma fácil cómo es la conexión entre las múltiples entidades del sistema. | | Una muestra de un diagrama de comunicación UML.  **Imagen:** 228125\_i24 |
| **Diagrama de tiempos:** muestra la conducta de cualquier tipo de objeto a través de una duración puntual. | | **Imagen:** 228125\_i25 |
| **Diagrama general interactivo:** reproduce el flujo de múltiples secuencias interactivas entre sí. Una de ellas es la actividad que está simbolizada por un margen que consta de varias unidades en su interior. | | **Imagen:** 228125\_i26 |

**3. Requerimientos de *software***

| Cuadro de texto |
| --- |
| Con respecto a los requerimientos de *software,* podemos decir que son las necesidades que tienen los *stakeholders* para requerir que el sistema deba cumplir de manera satisfactoria. En otras palabras, son los que definen las funciones que el sistema será capaz de realizar, describen los cambios que el sistema realiza sobre las entradas para producir salidas. Es de gran importancia que se describa el **qué** y no el **cómo** se deben hacer estos cambios. Estos requerimientos, con el tiempo que va avanzando el proyecto de *software*, se van convirtiendo en los algoritmos, la lógica y gran parte del código del sistema. |

Infografía interactiva Modal Requerimientos de *software*

| **Tipo de recurso** | Infografía interactiva Modal | |
| --- | --- | --- |
| **Texto introductorio** | Se pueden identificar principalmente tres niveles de requerimientos: | |
| El editable se encuentra en la carpeta Anexos con el nombre Infografia1.pptx | | |
| **Código de la imagen** | 228125\_i27 | |
| **Punto modal 1** | **Requerimientos de negocio:** estos requerimientos representan los objetivos establecidos por la organización, las necesidades de los usuarios y otros involucrados que están siendo afectados por el problema, sin descuidar las reglas del negocio de la organización. | **Requerimientos de negocio** |
| **Punto modal 2** | **Requerimientos de usuario:** son definiciones simples, en el lenguaje del usuario, que se usan para comunicar la solución de alto nivel a los afectados. | **Requerimientos de usuario** |
| **Punto modal 3** | **Requerimientos de sistema:** son los que definen de manera más clara la solución. Canalizan una o más características solicitadas por los usuarios en soluciones de *software* específicas. | **Requerimientos de sistema** |
| **Punto modal 4** | **Requerimientos funcionales:** especifican lo que el sistema debe hacer, es decir, describen acciones que el sistema debe ser capaz de realizar, sin pensar en las restricciones físicas. Los requerimientos funcionales detallan el comportamiento del sistema. | **Requerimientos funcionales** |
| **Punto modal 5** | **Requerimientos no funcionales:** especifican únicamente cualidades del sistema o atributos del ambiente del sistema. Constan de las siguientes características:   * Están relacionados en función al conjunto del sistema. * Son cualidades de las funciones que el sistema debe tener. Por consiguiente, no pueden existir requerimientos no funcionales sin que existan requerimientos funcionales. * El analista debe validar que exista un equilibrio entre estos requerimientos, porque es posible que se puedan producir contradicciones entre varios de estos. | **Requerimientos no funcionales** |
| **Punto modal 6** | **Requerimientos del producto:** diferencia el comportamiento del producto, como los requerimientos de desempeño, en cuánta memoria se requiere y la velocidad de ejecución del sistema. Por ejemplo:   * Funcionalidad * Fiabilidad * Usabilidad * Eficiencia * Mantenibilidad * Portabilidad | **Requerimientos del producto** |
| **Punto modal 7** | **Requerimientos organizacionales:** se derivan de procedimientos y de las políticas existentes en la organización del cliente y en la del equipo de desarrollado: estándares en los procesos que recomiendan usar, requerimientos de funcionamiento, como los lenguajes para programar y los requerimientos de entrega que determinan cuándo se entregará el producto final y su respectiva documentación. Por ejemplo:   * Requerimientos de entrega. * Requerimientos de implementación. * Requerimientos de estándares. | **Requerimientos organizacionales** |
| **Punto modal 8** | **Requerimientos externos:** se separan en su proceso de desarrollo y los factores externos al sistema. Incluyen los requerimientos de interoperabilidad que establecen la manera en que el sistema se relaciona con otros sistemas de la organización; los requerimientos legales que deben originarse para asegurar que el sistema trabaje dentro de la ley y los requerimientos éticos que son impuestos al sistema para garantizar que será aprobado por el usuario. Por ejemplo:   * Requerimientos de seguridad. * Requerimientos éticos. * Interoperabilidad. | **Requerimientos externos** |

**Especificación de Requisitos del *Software* (ERS)**

| Cuadro de texto |
| --- |
| Descripción del requisito de software descripción ilustración abstracta del vector de concepto. Descripción del sistema de software, herramienta ágil, análisis de negocios, especificaciones de desarrollo de proyectos, metáfora abstracta de documentos.  En cuanto al ERS, es un documento que su principal motivo es generar una descripción detallada de un producto de *software* a desarrollar, incluyendo su meta; los principales procesos de negocio que serán, como por ejemplo, comportamiento, características soportadas y parámetros clave de rendimiento. Por ende, sirve como una guía para el proceso y control de desarrollo, además, de esta forma se puede mantener a todos en el camino correcto.  Un ERS es generalmente aprobado al final de la fase de ingeniería de requisitos, la fase más temprana en el proceso de desarrollo de *software* y contiene requisitos funcionales y no funcionales. |

| **Tipo de recurso** | Slider Presentación | |
| --- | --- | --- |
| **Introducción** | No existen dos documentos ERS idénticos porque todos los proyectos de *software* son diferentes; algunos usando el modelo de desarrollo en cascada y otros practicando el desarrollo ágil. Sin embargo, es posible destilar los componentes principales de un ERS y crear un esquema aproximado sobre cómo debería ser: | |
| 1. Introducción   1.1 Propósito  1.2 Alcance  1.3 Definiciones, acrónimos y abreviaturas  1.4 Referencias  1.5 Visión global | |  |
| 2. Descripción general  2.1 Perspectiva del producto  2.2 Funciones del producto  2.3 Características del usuario  2.4 Restricciones  2.5 Supuestos y dependencias  2.6 Distribución de requisitos | |  |
| 3. Requisitos específicos  3.1 Requisitos funcionales  *3.1.1 Requisito funcional 1*  *3.1.1.1 Introducción*  *3.1.1.2 Entradas*  *3.1.1.3 Procesamiento*  *3.1.1.4 Salidas*  *3.1.2 Requisito funcional 2*  *3.1.3 Requisito funcional n*  3.2 Requisitos de interfaz externa  *3.2.1 Interfaces de usuario*  *3.2.2 Interfaces de hardware*  *3.2.3 Interfaces de software*  *3.2.4 Interfaces de comunicación*  3.3 Requisitos de ejecución  3.4 Requisitos de diseño  *3.4.1 Acatamiento de estándares*  *3.4.2 Limitaciones de hardware*  3.5 Atributos de calidad  *3.5.1 Seguridad*  *3.5.2 Mantenimiento*  3.6 Otros requisitos  *3.6.1 Base de datos*  *3.6.2 Operaciones*  *3.6.3 Adaptación de situación* | |  |

| Tipo de recurso | Cajón de texto de color |
| --- | --- |
| La primera etapa especifica el producto que se está desarrollando, su audiencia objetivo, su propósito, alcance y uso previsto. La segunda etapa brinda más información sobre las necesidades de los factores y los usuarios que podrían impedir que se cumplan los requisitos establecidos. La última etapa está dedicada a los requisitos específicos, tanto funcionales como no funcionales. | |

**Estándar IEEE 830**

| Cuadro de texto |
| --- |
| ¿Sabía que uno de los aspectos más importantes para la implementación de un *software* es tener claros los requerimientos del usuario final? Por esto se ha visto la necesidad de plantear una metodología que describa los métodos y enfoques recomendados para definir un ERS, ayudando a los clientes de *software* a detallar con precisión lo que desean obtener y facilitando al mismo tiempo, a todos los proveedores, la comprensión para validar exactamente lo que realmente el cliente desea. |

| **Tipo de recurso** | | Pestañas o tabs Verticales |
| --- | --- | --- |
| **Introducción** | | Este estándar establece que las funciones deberán expresarse como una jerarquía funcional, que a su vez permite organizar esta subsección de múltiples formas y sugiere, entre otras, las siguientes: |
| Icono de concepto de agrupamiento. Técnica de minería de datos abstracta idea ilustración de línea delgada. Segmentación jerárquica de datos. Dibujo de contorno aislado. Trazo editable. Fuentes Arial, Myriad Pro-Bold utilizadas  **Imagen:** 228125\_i28 | | |
| **Por tipo de usuario** | Distintos usuarios poseen distintos requerimientos. Para cada clase de usuario que exista en la organización, se especificarán los requerimientos funcionales que le afecte o tenga mayor relación con sus tareas. | |
| **Por objetos** | Son entidades del mundo real que serán mostradas en el sistema. Por cada objeto, se describen sus funciones y sus atributos. Los objetos pueden agruparse en clases. | |
| **Por objetivos** | Es un servicio que se aspira a que ofrezca el sistema y que requiere una cantidad determinada de entradas para generar su resultado. Para cada objetivo/subobjetivo que se siga con el sistema, se describen las funciones que permitan llevarlo a cabo. | |
| **Por estímulos** | Se especifican los posibles estímulos que recibe el sistema y las funciones relacionadas con dicho estímulo. Por jerarquía funcional: si ninguna de las anteriores alternativas resulta de ayuda, la funcionalidad del sistema se especifica como una jerarquía de funciones que comparten entradas, salidas o datos internos. Se detallarán las funciones y las subfunciones del sistema. | |

**Priorización de requerimientos**

| Cuadro de texto |
| --- |
| Ahora, durante la administración de requisitos de un desarrollo de un producto/servicio, además de ejecutar una correcta escritura de los requisitos y recogida, alcanza una gran importancia generar una priorización de los requisitos. Este tipo de tarea no es nada fácil, debido a que se pueden encontrar intereses diferentes.  Una de las técnicas más usadas es conocida como la técnica MoSCoW. La cual da priorización al escoger los requisitos, basada en el hecho de que, aunque todos los requisitos se estiman que son importantes, es primordial destacar aquellos requisitos principales y que aportan un mayor valor agregado al negocio y son considerados indispensables, de forma que el producto/servicio no se puede colocar en producción si incumple alguno de estos requisitos; esto ayudará a guiar el desarrollo de forma más eficiente. |

| **Tipo de recurso** | Infografía interactiva Punto caliente | |
| --- | --- | --- |
| **Texto introductorio** | La técnica MoSCoW no determina si un requisito es de prioridad alta, media o baja. Esta técnica ayuda a dividir los requisitos en cuatro grandes categorías: | |
| Realizar una imagen como la que se presenta. Los textos editables se encuentran en la carpeta Anexos con el nombre Infografia2.pptx | | |
| **Código de la imagen** | 228125\_i29 | |
| **Punto caliente 1** | Requisitos obligatorios y fundamentales para amortizar las necesidades del negocio. Si estos no se cumplen, se verá afectado el éxito del servicio. | **Must** |
| **Punto caliente 2** | Requisitos que deberían efectuarse en la medida de lo posible. El éxito del proyecto/servicio no dependerá de primera mano del cumplimiento de estos requisitos. Estos requisitos pueden ser satisfechos mediante resultados temporales, o llegado el caso si fuera necesario, podrían ser reemplazables si existiera alguna causa que lo justificara. | **Should** |
| **Punto caliente 3** | Requisitos que son realmente interesantes para que cumpla un servicio. Se trata de requisitos suplementarios que se implementarán en el caso de disponer de presupuesto para ello y tiempo. Estos requisitos mejorarían la productividad del servicio pero podrían ser eliminados de forma fácil. | **Could** |
| **Punto caliente 4** | Requisitos que se han tomado la decisión de no implementar de momento, pero que serán tomados en cuenta en un futuro con la meta de mejorar el servicio/producto. | **Won’t** |

| Tipo de recurso | Cajón de texto de color |
| --- | --- |
| Estas técnicas nos ayudan a definir una “línea roja”, reconociendo aquellos requisitos que son obligatorios y que deben ser incluidos en el desarrollo del producto. | |

| **Tipo de recurso** | Acordeón tipo 1 |
| --- | --- |
| **Introducción** | Una vez que se tengan identificadas las clasificaciones de requerimientos que existen y las características que deben cumplir, se puede comenzar con la descripción de las actividades que ayudarán a realizar una buena obtención de requerimientos y para ello se deben seguir los siguientes pasos: |
| Especificaciones de desarrollo de proyectos conjunto de ilustraciones vectoriales de concepto abstracto. Análisis de negocio, descripción de requisitos de software, documento de visión y ámbito, análisis SWOT, metáfora abstracta de caso de usuario.  **Imagen:** 228125\_i30 | |
| **Obtención de requerimientos**  Es la fase inicial de cada ciclo. Los analistas de requerimientos deben trabajar en conjunto con el cliente con el fin de encontrar el problema que el sistema debe solucionar. Este proceso lo regula una junta llamada *kick off*, donde se debe testimoniar lo siguiente:   * Objetivo del sistema, fechas provisionales del inicio y fin de la ejecución del proyecto. * Presentación del grupo de trabajo. * Presentación de *stakeholders*. * Fechas provisionales de reuniones con el cliente (solo aplica siempre y cuando sea consultoría). | |
| **Análisis de requerimientos**  Es el segundo paso, implica analizar, refinar y escudriñar/examinar los requerimientos obtenidos para asegurar que todos los clientes involucrados entienden lo que pidieron; para encontrar errores, omisiones y otras deficiencias.  Las actividades a contemplar durante esta etapa son:   * Reducir confusiones en los requerimientos: se realizan las actividades que permiten borrar los términos que tienen más de una acepción. * Traducir a lenguaje técnico los requerimientos: los requerimientos, después de la etapa anterior, deben ser tratados para llevarlos a un lenguaje de comunicación lo más técnico posible. * Proponer un modelo lógico: se debe edificar un modelo del problema, sea en términos de diagramas o cualquier otro tipo de representación que se considere viable para el modelado. | |
| **Especificación de requerimientos**  En esta fase se realiza la documentación de los requerimientos concertados con el cliente, en un nivel necesario de detalle; se documenta la descripción de manera completa de las funcionalidades del sistema y las necesidades que serán desarrolladas; de esta manera, también describe el alcance del sistema y la forma como hará sus funciones, definiendo los requerimientos funcionales y no funcionales. | |
| **Verificación de requerimientos**  Se recomienda seleccionar varios revisores de diferentes ámbitos disciplinares, puede ser un arquitecto, un analista, etc.; pero debe ser alguien que conozca la ingeniería de requerimientos, con el fin de realizar validaciones al documento, aplicarles pruebas de escritorio, etc. | |
| **Aceptación de requerimientos**  Este es un proceso donde los analistas implicados se reúnen con el cliente y comienzan a dar una validación de manera formal al documento, de esta manera, comienzan a leer y explicar cada requerimiento; esto con el fin de que todo el grupo comprenda lo que se realizará para cada requerimiento. Una vez que todos estén sincronizados y de acuerdo, se hace la aprobación/aceptación de los detalles de requerimientos, se realiza un acuerdo formal que lo contenga y según las respectivas especificaciones documentadas, será con base en eso que se construya y se pida al cliente una aprobación formal vía correo electrónico o una firma sobre el documento físico. | |
| **Administración de requerimientos**  Se realiza la etapa de todo el proyecto, esto implica llevar un control de los cambios, asegurándose de hacer ver al cliente el impacto en tiempo y costo de entrega del proyecto. Otro punto importante es que debe asegurarse de que todas las actividades del proyecto se den a tiempo para no causar retrasos en la entrega. | |

**4. Técnicas de documentación de requerimientos en proyectos ERS**

| Cuadro de texto |
| --- |
| El objetivo principal de la Especificación de Requisitos del Sistema (ERS) es ser el intermediario como medio de reproducción entre clientes, usuarios, ingenieros de requisitos y desarrolladores. |

| **Tipo de recurso** | Pestañas o tabs horizontales | |
| --- | --- | --- |
| **Introducción** | Las técnicas para analizar los requerimientos expuestas a continuación, parten de la matriz de trazabilidad de requerimientos del proyecto/especificación de requisitos. | |
| **Descomposición funcional** | Consiste en agarrar los requerimientos de *software*, separarlos en partes y examinarlos individualmente. De ser necesario, se pueden fragmentar en muchas más partes hasta generar un nivel adecuado de detalle. En cierto modo, el proceso es semejante al de la creación de una estructura de desglose de trabajo de un proyecto. En ingeniería de sistemas, la descomposición funcional se basa en definir un sistema en términos funcionales, para luego generar funciones de bajo nivel y formalizar las relaciones con respecto a las funciones de nivel alto. Esto con el fin de dividir un sistema de tal manera que cada componente se pueda explicar sin necesidad de llamar a otro componente. De esta manera, cada parte del sistema consta de funciones independientes que pueden reemplazarse y rehusarse. | Imagen: 228125\_i31 |
| **Especificación vía sentencias textuales** | El procedimiento consiste en agarrar el requerimiento de un producto del levantamiento de información, para generar una narrativa con mayor detalle. En este caso, no se usan herramientas visuales como los casos de uso o flujogramas, es simplemente una explicación con mayor detalle del requerimiento en lenguaje natural. | Icono de concepto de texto. Diferentes tipos de ideas de información textual ilustración de línea delgada. Libros, noticias, anuncios. Documentos y archivos, artículos. Dibujo de contorno aislado del vector. Trazo editable  Imagen: 228125\_i32 |
| **Modelado del proceso** | Engloba la creación de diagramas de flujo de procesos a través de los requerimientos del *software*. Existen muchas herramientas de modelado de procesos, cada una de ellas con sus propias reglas y símbolos. Los procesos se inician por eventos y pueden incluir actividades manuales, automatizadas o combinación entre ambas; cuando los procesos son grandes y complejos, es necesario desglosarlos en componentes (subprocesos). La **relación entre la gerencia de proyectos y los diagramas de flujo** es necesaria para el éxito. | Modelado de Procesos: Cómo obtener excelentes resultados en poco tiempo |  Bitácora de Análisis de Negocio  Imagen: 228125\_i33 |
| **Modelado de dominio** | Consiste en examinar el *software* o sistemas relacionados en un dominio, con el fin de descubrir las partes comunes y las partes que los diferencian. Un modelo de dominio abarca diagramas conceptuales que comprenden tanto el comportamiento de un sistema como el de sus datos. Un tipo de modelo de dominio son los diagramas de funcionalidades, que representan de forma “compacta” los sistemas o aplicaciones en términos de sus características. | Imagen: 228125\_i34 |
| **Caso de uso** | Es una secuencia de intercambio entre un sistema y/o un servicio. Además, es una de las técnicas con mayor expansión para detallar el comportamiento de un sistema. También proporciona un formato estructurado y simple que puede ser divulgado entre desarrolladores y usuarios, además de implementarse para el análisis de los requerimientos de *software*. Son importantes en sistemas informáticos basados en la funcionalidad, que se van a ejecutar en sistemas orientados a objetos y con UML. | Imagen: 228125\_i35 |
| ***Checklist*** | También conocida como la lista de chequeo, consiste en una serie de revisiones/preguntas que se hacen sobre los **requerimientos de s*oftware***y que son mostradas de forma escrita. Por ejemplo:   * ¿Se han realizado consideraciones de seguridad? * ¿Se han determinado los requisitos de *software* y *hardware*? * ¿Se ha agregado el código de referencia para el reconocimiento del requisito en el desglose de requerimientos? * ¿El requerimiento es único? (No existe duplicidad con otro requerimiento). * ¿Está escrito el requerimiento en un lenguaje claro y conciso?   La lista de chequeo sirve como marco de trabajo y técnicas para revisar el requerimiento, simplificando su análisis de forma estructurada. Los requerimientos se pueden verificar sobre la definición del alcance o **matriz de trazabilidad de requerimientos**. | Icono de línea del Portapapeles. Símbolo de signo de lista de comprobación para diseño de sitio web y aplicación.  Imagen: 228125\_i36 |
| **Inspección** | Es la revisión no invasiva de los requerimientos de *software*. Por ejemplo:   * Validar un *software* visualmente para corroborar que las pantallas solicitadas por el cliente se encuentran incluidas en el desarrollo. * Verificar la entrada de los campos necesarios para el ingreso de datos. * Verificar la efectividad de los botones para empezar la funcionalidad que ha sido solicitada. * Verificar que el requerimiento cumpla con los estándares definidos para la aplicación. | Control de calidad QA de garantía de calidad, mejor estándar, satisfacción del producto. Inspección para errores o errores. Pruebas de software. Ilustración vectorial de caricatura plana con personajes para banner web  Imagen: 228125\_i37 |
| **Prototipos** | Consiste en generar una representación visual (interfaz gráfica con el usuario) de los requerimientos de *software*. Es una herramienta importante para validar con los clientes, usuarios e interesados de un proyecto debido a que el diseño debe corresponder con los requerimientos de *software*. Además, permite a usuarios y desarrolladores comprender mejor los requerimientos, determinar cuáles son deseables y cuáles indispensables, de esta manera poder identificar los riesgos de forma temprana. La creación de prototipos conlleva a generar mucha comunicación entre usuarios y desarrolladores. | Diseñadores de creación de sitios web de dibujo ux desarrollo de aplicaciones. Concepto de experiencia del usuario.  Imagen: 228125\_i38 |

| Cuadro de texto |
| --- |
| Para lograr un proyecto de *software* exitoso es necesario comprender el ámbito del trabajo a ejecutar, los riesgos en que pueda llegar a incurrir, los recursos necesarios para su ejecución, las actividades a llevar a cabo, el costo de los presupuestos controlados y un plan a seguir como soporte. |

| **Tipo de recurso** | Acordeón tipo 1 |
| --- | --- |
| **Introducción** | Lo siguiente es ver cuáles son las características de una buena documentación ERS y qué parámetros deben cumplir estos requerimientos en un proyecto, los cuales se presentan a continuación. |
| Descripción del requisito de software descripción ilustración abstracta del vector de concepto. Descripción del sistema de software, herramienta ágil, análisis de negocios, especificaciones de desarrollo de proyectos, metáfora abstracta de documentos.  **Imagen:** 228125\_i39 | |
| **Necesario:** si se tiene alguna inquietud acerca de la necesidad del requerimiento, se pueden preguntar: ¿cuál sería el peor caso al no incluir cierto requerimiento? Si no se obtiene una respuesta, entonces es probable que no sea un requerimiento obligatorio. | |
| **Completo:** un requerimiento está totalmente finalizado si no necesita ampliar detalles en su descripción, es decir, si se suministra la información suficiente para su entendimiento. | |
| **Consistente:** es consistente siempre y cuando no sea contradictorio con otro requerimiento. | |
| **Correcto**: estar de acuerdo entre las dos partes, es decir, contiene una única idea. | |
| **Factible**: el requerimiento deberá ser totalmente viable y estar dentro de un calendario, presupuesto y otras restricciones; si existe alguna duda de su factibilidad, hay que volver a investigar, generar sus respectivas pruebas de concepto para saber su grado de factibilidad y complejidad, y si aún así el requerimiento es no viable, hay que volver a replantear el requerimiento. | |
| **Modificable**: que se pueda ajustar con el tiempo. | |
| **Priorizado:** organizar el requerimiento nos ayuda a saber el grado de necesidad de este, si es crítico o esencial. | |
| **Verificable:** si el requerimiento no se puede validar, entonces, ¿cómo se puede saber si el requerimiento se cumplió o no? Debe ser posible validarlo, sea por análisis de prueba o demostración e inspección. Cuando se esté generando un requerimiento, se deberán decidir los criterios de aceptación pertenecientes al requerimiento. | |
| **Rastreable:** el requerimiento y la funcionalidad se deben ordenar de tal forma que cada función se pueda rastrear, lo que facilita la validación del diseño y las pruebas. | |
| **Claro:** un requerimiento debe ser claro, fácil de leer y entender; su composición debe ser simple para aquellos que vayan a averiguarlo en un futuro. | |

| Cuadro de texto |
| --- |
| Es así como el uso de este tipo de estándares implica un grupo de ventajas, como, por ejemplo:   * La sencillez en la lectura: esto se debe a que una vez que los lectores han comprendido y entendido la plantilla, el interpretar el resto de los requisitos es mucho más fácil y el tiempo usado para esto es mucho menor que si no se utilizan. * La escritura: una vez reconocida la información que obtiene dicha plantilla, la escritura es mecánica, por ende, la probabilidad de que se olvide algo de información es menor y la revisión de los requisitos es mucho más sencilla. * Las personas encargadas o revisores, al inspeccionar el documento, pueden utilizar el estándar con el fin de validar si alguna sección ha quedado incompleta y de esta manera llevar el proceso para que sea revisado. * Los requisitos son más fáciles de recoger.   Por otro lado, la desventaja de que el *software* es intangible, ¿qué significa? Que el *software* no puede ser tocado, lo que lleva a que, en el caso hipotético de darse algún problema, los efectos de este no son visibles de una manera sencilla, sino que se confían de otros para hacer la documentación oportuna para revisar el progreso. |

| Cuadro de texto |
| --- |
| Recuerde explorar los demás recursos que se encuentran disponibles en este componente formativo; para ello, diríjase al menú principal donde encontrará la síntesis, una actividad didáctica para reforzar los conceptos estudiados, material complementario, entre otros. |

**SÍNTESIS**

| **Tipo de recurso** | Síntesis |
| --- | --- |
| Desarrollo de aplicaciones web full stack  Síntesis: Construcción del informe de requisitos | |
| **Introducción** | El siguiente mapa integra los criterios y especificidades de los conocimientos expuestos en el presente componente formativo. |
|  | |

**ACTIVIDAD DIDÁCTICA**

| **Tipo de recurso** | Actividad didáctica. Opción múltiple | |
| --- | --- | --- |
| Apreciado aprendiz: a continuación, encontrará una serie de preguntas que deberá resolver con el objetivo de evaluar la aprehensión de los conocimientos expuestos en este componente formativo. | | Desarrollo de tecnologías de diseño y codificación de sitios web de desarrollo de equipos de programadores que trabajan en la oficina de la empresa de software  **Imagen:** 228125\_i41 |
| Las metodologías de desarrollo de *software* ágiles son las que: | |  |
| No permiten realizar cambios y son poco flexibles. | | No permiten realizar cambios y son bastante flexibles. |
| Permiten realizar cambios y son bastantes flexibles. (correcta) | | Permiten realizar cambios y son poco flexibles. |
| Retroalimentación positiva:  Muy bien, estas metodologías son las que se basan en la metodología incremental, donde en cada ciclo de desarrollo se van añadiendo nuevas funcionalidades al desarrollo final.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre metodologías de desarrollo de *software* ágiles. | | |
| Las principales metodologías ágiles son: | |  |
| Kanban, Scrum, Lean y Waterfall. | | Waterfall, Prototipado, Espiral, Incremental y Programación extrema (XP). |
| Waterfall, Prototipado, Espiral, Incremental y Diseño Rápido de Aplicaciones (RAD). | | Kanban, Scrum, Lean y Programación extrema (XP). (correcta) |
| Retroalimentación positiva:  Muy bien, las principales metodologías ágiles son: Kanban (esta metodología consiste en separar las actividades en fragmentos mínimos y organizarlas en un tablero de trabajo, separándolas en tareas pendientes, en curso y finalizadas); Scrum (metodología incremental que divide las tareas y requisitos de forma similar a Kanban); Lean (está configurado para que grupos pequeños de desarrollo altamente capacitados puedan elaborar cualquier tarea en poco tiempo) y Programación extrema (su principal objetivo es generar un buen ambiente de trabajo con el grupo y que exista una retroalimentación constante del cliente).  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre metodologías ágiles. | | |
| Una de las siguientes opciones es una ventaja del uso de lenguaje unificado de modelado (UML). | |  |
| Es un recurso adicional que ayuda al desarrollador. | | Ayudan a visualizar cómo es o cómo se quiere que sea el sistema. (correcta) |
| Generan retrasos, por eso no son buenos de usar. | | Ninguna de las anteriores. |
| Retroalimentación positiva:  Muy bien, las ventajas del uso de UML son: ayudar a analizar el cómo se implementa, facilitar la comunicación entre el grupo al existir un lenguaje en común, disponer de documentación que trasciende al proyecto, ayudar a visualizar cómo es o cómo se quiere que sea el sistema, especificar el comportamiento y la estructura del sistema, guiar la construcción del sistema.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre las ventajas del uso de UML. | | |
| Los diagramas de máquina de estado pertenecen al: | |  |
| Diagrama provisional del UML. | | Diagrama general de UML. |
| Diagrama UML de comportamiento. (correcta) | | Diagrama estructural UML. |
| Retroalimentación positiva:  Muy bien, los diagramas UML de comportamiento son diagramas que determinan la funcionalidad general de cada unidad y el comportamiento del sistema. En los diagramas UML de comportamiento se encuentran el diagrama de máquina de estado, el diagrama de actividad, el diagrama de casos de uso, el diagrama secuencial, el diagrama de comunicación, el diagrama de tiempos y el diagrama general interactivo.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre los diagramas UML de comportamiento. | | |
| En la clasificación de requerimientos se pueden identificar principalmente tres niveles: | |  |
| Requerimientos de producto, requerimientos organizacionales y requerimientos externos. | | Requerimientos de negocio, requerimientos de usuario y requerimientos externos. |
| Requerimientos organizacionales, requerimientos de usuario y requerimientos de sistema. | | Requerimientos de negocio, requerimientos de usuario y requerimientos de sistema. (correcta) |
| Retroalimentación positiva:  Muy bien, en la clasificación de requerimiento se identifican tres niveles: requerimientos de negocio, requerimientos de usuario y requerimientos de sistema.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre la clasificación de requerimiento. | | |
| Documento cuyo propósito es proporcionar una descripción completa de un producto de *software* a desarrollar, incluyendo su propósito, los principales procesos de negocio que serán soportados, características, parámetros clave de rendimiento y comportamiento. | |  |
| Requerimientos no funcionales. | | Requerimientos funcionales de *software*. |
| Recopilación de requisitos de *software.* | | Especificación de requisitos del *software* (ERS). (correcta) |
| Retroalimentación positiva:  Muy bien, un ERS es un documento que sirve como un mapa de guía para el proceso de desarrollo y de esta manera, manteniendo a todos en el camino correcto, generalmente es aprobado al final de la fase de ingeniería de requisitos, la fase más temprana en el proceso de desarrollo de *software;* y contiene requisitos funcionales y no funcionales.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre ERS. | | |
| Los requerimientos funcionales son los que: | |  |
| Detallan el comportamiento del sistema. (correcta) | | Especifican únicamente cualidades del sistema o atributos del ambiente del sistema. |
| Representan los objetivos establecidos por la organización, las necesidades de los usuarios y otros involucrados que están siendo afectados por el problema, sin descuidar las reglas del negocio de la organización. | | Diferencian el comportamiento del producto. |
| Retroalimentación positiva:  Muy bien, los requerimientos funcionales son los que especifican lo que el sistema debe hacer, es decir, describen acciones que el sistema debe ser capaz de realizar sin pensar en las restricciones físicas, o sea, detallan el comportamiento del sistema.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre los requerimientos funcionales. | | |
| Según la técnica de MoSCoW, a cuál de las palabras pertenece la siguiente definición: los requisitos deberían efectuarse en la medida de lo posible. | |  |
| Should (correcta) | | Must |
| Could | | Won’t |
| Retroalimentación positiva:  Muy bien, la técnica MoSCoW no determina si un requisito es de prioridad alta, media o baja, ayuda a dividir los requisitos en cuatro grandes categorías: Must (requisitos obligatorios y fundamentales para amortizar las necesidades del negocio); Should (requisitos que deberían efectuarse en la medida de lo posible); Could (requisitos que son realmente interesantes para que cumpla un servicio) y Won’t (requisitos que se han tomado la decisión de no implementar de momento, pero que serán tomados en cuenta en un futuro con la meta de mejorar el servicio/producto).  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre la técnica MoSCoW. | | |
| Las actividades que se contemplan durante la etapa de análisis de requerimientos son: | |  |
| Necesario, completo, consistente, correcto, factible, modificable, priorizado, verificable, rastreable y claro. | | Descomposición funcional, especificación vía sentencias textuales, modelado del proceso, modelado de dominio, caso de uso y *checklist*. |
| Reducir confusiones en los requerimientos, traducir a lenguaje técnico los requerimientos, proponer un modelo lógico, especificación de requerimientos, verificación de requerimientos, aceptación de requerimientos y administración de requerimientos. (correcta) | | Ninguna de las anteriores. |
| Retroalimentación positiva:  Muy bien, el análisis de requerimientos es el segundo paso, implica analizar, refinar y escudriñar/examinar los requerimientos obtenidos para asegurar que todos los clientes involucrados entienden lo que pidieron, y para encontrar errores, omisiones y otras deficiencias.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre el análisis de requerimientos. | | |
| Consiste en examinar el *software* o sistemas relacionados en un dominio, con el fin de descubrir las partes comunes y las partes que los diferencian. La anterior definición pertenece a: | |  |
| Modelado de dominio (correcta) | | Inspección |
| *Checklist* | | Prototipos |
| Retroalimentación positiva:  Muy bien, el modelado de dominio consiste en examinar el *software* o sistemas relacionados en un dominio, con el fin de descubrir las partes comunes y partes que los diferencian; y abarca diagramas conceptuales que comprenden tanto el comportamiento de un sistema como el de sus datos.  Retroalimentación negativa:  ¡Incorrecto! Es necesario revisar nuevamente el tema sobre el modelado de dominio. | | |

**Retroalimentación general positiva:**

¡Felicitaciones! Ha logrado una óptima aprehensión de los conocimientos relacionados con la construcción del informe de requisitos.

**Retroalimentación general negativa:**

¡Inténtelo de nuevo! Lo invitamos a revisar nuevamente el material de estudio para afianzar los conocimientos presentados. ¡Ánimo!

**MATERIAL COMPLEMENTARIO**

| Tipo de recurso | Material complementario | | |
| --- | --- | --- | --- |
| Tema | Referencia APA del material | tipo | Enlace |
| Metodologías de desarrollo de *software* | OS Infor. (2019). *Metodologías de desarrollo de software – Análisis y diseño* (video). YouTube. <https://www.youtube.com/watch?v=5tyU_h69F-s> | Video | <https://www.youtube.com/watch?v=5tyU_h69F-s> |
| Lenguaje unificado de modelado | Lucid Software Español. (2019). *Tutorial – Diagrama de Clases UML* (video). YouTube. <https://www.youtube.com/watch?v=Z0yLerU0g-Q&t=164s> | Video | <https://www.youtube.com/watch?v=Z0yLerU0g-Q&t=164s> |
| Requerimientos de *software* | Ingeniería de Software de Élite. (2020). *01-Ingeniería de Requerimientos ¿Qué son los requerimientos de software?* (video). YouTube. <https://www.youtube.com/watch?v=Hedi-Qobgbc> | Video | <https://www.youtube.com/watch?v=Hedi-Qobgbc> |
| Técnicas de documentación de requerimientos en proyectos ERS | Varon, A. (2021). *Como crear un documento de Especificación de Requerimientos de Software* (video). YouTube. <https://www.youtube.com/watch?v=AotyBHVKp8I> | Video | <https://www.youtube.com/watch?v=AotyBHVKp8I> |

**GLOSARIO**

| **Tipo de recurso** | Glosario |
| --- | --- |
| Accesibilidad: | capacidad por la cual un *software* facilita el mantenimiento de sus componentes. |
| Adaptabilidad: | facilidad con la que un *software* se adapta a diferentes condicionantes del sistema y necesidades del usuario. |
| Base de datos: | conjunto de datos relacionados que van a ser utilizados por un conjunto de programas. |
| Calidad del *software:* | conjunto de especificaciones de un producto *software*, relacionado con su capacidad de satisfacer unas necesidades dadas. |
| Caso de prueba: | conjunto de datos de prueba y los procedimientos asociados, desarrollados para un determinado caso, tal como probar un camino particular de un programa o verificar el cumplimiento de un requisito específico. |
| Ciclo de vida del *software:* | periodo de tiempo que empieza cuando se crea un producto *software* y finaliza cuando el producto pierde su utilidad. |
| Datos: | representación de los hechos, instrucciones de una manera satisfactoria para la comunicación. |
| Diagramas UML de comportamiento: | determinan la funcionalidad general de cada unidad y el comportamiento del sistema. |
| Estándar: | requisitos obligatorios empleados y aplicados a dirigir un enfoque uniforme para el desarrollo de *software*. |
| Trazabilidad: | habilidad para seguir la vida de un requerimiento en ambos sentidos, desde sus orígenes o hasta su implementación por medio de las especificaciones propuestas durante el proceso de desarrollo. |

**REFERENCIAS BIBLIOGRÁFICAS:**

| **Tipo de recurso** | Bibliografía |
| --- | --- |
| Kishorekumar 62. (s.f.). *Domain model.* <https://commons.wikimedia.org/wiki/File:Domain_model.png> | |
| Letelier, P. y Penadés, M. C. (2006). *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).* <https://www.researchgate.net/publication/26428496_Metodologias_agiles_para_el_desarrollo_de_software_eXtreme_Programming_XP> | |
| López, C. (2017). *Efectividad metodológica para el levantamiento de requerimientos de una aplicación web que permita realizar el proceso de planeación de la acción pedagógica* (Tesis maestría, Universidad Autónoma de Bucaramanga en convenio con Universitat Oberta de Catalunya). Repositorio UNAB. <https://repository.unab.edu.co/bitstream/handle/20.500.12749/3419/2017_Tesis_Lopez_Caballero_Cesar_Augusto.pdf?sequence=1> | |
| Marijo TM. (2016). *Diagrama de despliegue.* <https://commons.wikimedia.org/wiki/File:Diagrama-de-despliegue.jpg> | |
| Microsoft. (2021). *Crear un diagrama de comunicación UML.* <https://support.microsoft.com/es-es/office/crear-un-diagrama-de-comunicaci%C3%B3n-uml-911956f4-5f19-4a58-97a3-bb14110a5ed1> | |
| Pressman, R. (s.f.). *Ingeniería del software. Un enfoque práctico.* <http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF> | |
| Sarmiento, P. y Hernández, D. (2017). *Metodología para la optimización de los procesos de recolección de información y análisis en la etapa de especificación de requerimientos de software.* (Trabajo final, Universidad Distrital Francisco José de Caldas). Repositorio Universidad Distrital. <https://repository.udistrital.edu.co/bitstream/handle/11349/7804/METODOLOG%C3%8DA%20PARA%20LA%20OPTIMIZACI%C3%93N%20DE%20LOS%20PROCESOS%20DE%20RECOLECCI%C3%93N%20DE%20INFORMACI%C3%93N%20Y%20AN%C3%81LISIS%20EN%20LA%20ETAPA%20DE%20ESPECIFICACI%C3%93N%20DE%20REQUERIMIENTOS%20DE%20SOFTWARE.pdf?sequence=1&isAllowed=y> | |
| Stkl. (2015). *Umlprofildiagramm.* <https://fr.wikipedia.org/wiki/Fichier:UmlProfildiagramm-1.svg> | |
| Wilfredor. (2010). *Diagrama de clases.* <https://commons.wikimedia.org/wiki/File:Diagrama_de_clases.svg> | |
| Yohandry14. (2012). *Ejemplo de diagrama de componentes.* <https://commons.wikimedia.org/wiki/File:Ejemplo_de_diagrama_de_componentes.png> | |