**Datos de identificación del programa de formación**

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

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501095 -Diseño de los artefactos necesarios para la solución de *software* | RESULTADOS DE APRENDIZAJE | 220501095-01 - Modelar los requerimientos de acuerdo con una de las metodologías de desarrollo existentes. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 6 |
| NOMBRE DEL COMPONENTE FORMATIVO | Análisis, diseño y arquitectura de *software* |
| BREVE DESCRIPCIÓN | En este componente formativo se tratarán los temas de metodologías de desarrollo de *software* tradicional, ágil y elementos de estructura arquitectónica. |
| PALABRAS CLAVE | UML, RAD, clases, objetos |

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

# **Tabla de contenido**

1. **Metodologías de desarrollo de *software* tradicionales**
2. **Metodologías de desarrollo de *software* ágiles**
3. **Elementos del modelo objetos**
4. **Elementos de la estructura arquitectónica**

**Introducción**

|  |
| --- |
| Cuadro de texto |
| Bienvenido a este componente formativo, donde el contenido a estudiar es muy importante para gestar las buenas prácticas de diseño de un producto *software*; por eso, se le invita a revisar ―antes de empezar― el video de introducción, así tendrá más claridad sobre la temática que estudiará. |

**Guion de video introductorio**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de recurso** | Video animación 2D | | | |
| **NOTA** |  | | | |
| **Título** | Introducción | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración** | **Texto** |
| **1** | Programador desarrollador teclear lenguajes de código fuente de script codificar símbolos de programación de datos de proyectos ingeniería de software de TI informática computadora. 3 quinquies de representación.  Diverso equipo de ingenieros con portátil y tableta analiza y discute cómo funciona un brazo robótico futurista y mueve un objeto metálico. Están en un laboratorio de investigación de alta tecnología con equipos modernos.  Tecnología digital, concepto de desarrollo de software. Programador de codificación, ingeniero de software trabajando en laptop con placa de circuito y javascript en pantalla virtual, internet de las cosas IoT  Controlador técnico trabajando en su estación de trabajo con múltiples pantallas. Muestra La Información Técnica. Está solo en el Centro de Control de Sistemas. |  | Los proyectos de *software* están conformados por complicados procesos que involucran a diferentes personajes y conocimientos, como: miembros del equipo, ingeniería de *software*, sistemas de información y procedimientos que incluyen reunir requisitos, pruebas y mantenimiento acordes con metodologías que ayudan a concretar el producto planificado. | Proyectos de *software*  Miembros del equipo  Ingeniería de *software*  Sistemas de información |
| **2** | Sistema informático hackeado. Pantalla de software de virus en el monitor  La imagen debe ser similar pero que no diga hackeado, sino que tiene una crisis de programación. |  | En la industria del *software* con frecuencia los proyectos fracasan. Inclusive se experimentó una crisis de *software* que se debe a las malas prácticas en el proceso de su desarrollo o la gestión del proyecto que se diseña para satisfacer necesidades del cliente, pero esta tecnología se encuentra en cambio y avance permanentes y son la experiencia y la práctica las que permiten administrarlos de manera eficiente. | Crisis de *software*  Malas prácticas  Gestión del proyecto |
| **3** | Joven asiática liderando un equipo creativo de negocios en el proyecto de diseño de software de aplicaciones móviles. Reunión de ideas, trabajo conjunto, tecnología de Internet, poder de las niñas, concepto de trabajo en equipo de compañeros de oficina  Concepto de Scrum Framework. Diagrama de procesos de Scrum como un programa de desarrollo de software ágil. Equipo discutiendo tareas en la pizarra Kanban y trabajando en la laptop. Reunión de negocios. Ciclo de trabajo de administración de proyectos |  | Este componente formativo brinda conocimientos fundamentales sobre metodologías tradicionales y ágiles en desarrollo de *software* para realizar una adecuada comprensión sobre sus conceptos, tipos, características principales y forma de implementarse en un ambiente empresarial y productivo. | Metodologías tradicionales y ágiles  Conceptos  Tipos  Características  implementación |
| **4** | Gran juego de desarrollo de sitios web. Sitio Web que establece pasos, planificación de proyectos de TI. Programación de páginas Web y creación de una interfaz receptiva. Ilustración plana aislada | Colocar si habrá un sonido o música de fondo | Es importante tener en cuenta que en la industria del *software* es vital tener claridad de lo que se va a desarrollar, la certeza de que el cliente está de acuerdo y, sobre todo, saber qué es lo que se quiere para plasmarlo de la forma correcta. | Claridad  Certeza |
| **5** | Terapia de grupo en sesión sentada en círculo alto en una sala luminosa  sistema de desarrollo de software de modelado de equipo de lenguaje unificado Uml  Ingeniero De Software Dibujando Un Diagrama De Caso De Uso De Uml  diagrama UML. Usar ilustración vectorial de modelo de caso. |  | En un proceso de diseño de *software* la comunicación es muy importante; por eso las formas correctas de llevarla a cabo son definitivas. En este componente se brindan las herramientas fundamentales a través del lenguaje UML y de gráficos o diagramas para que su comunicación sea asertiva, sin dejar nada a la incertidumbre. Se detallarán los diagramas vitales para el diseño de *software,* como el diagrama de clases que indica la estructura del sistema y permite que en el equipo de desarrolladores se hable el mismo idioma. | Comunicación  Lenguaje UML  Diagrama de clases |
| **6** | Concepto de desarrollo web en estilo plano. El equipo de desarrolladores construye la escena de la aplicación web. Desarrollo completo de pila, ingeniería de software, diseño y programación. Ilustración vectorial con caracteres de personas  Aplicación con Tecnología J2EE para la Administración y control en el Área  de Logística (página 2) |  | Se utilizará un carácter teórico-práctico; por esto se entiende que se llevará una secuencia de teorías muy importantes para la conceptualización de los conceptos sobre las metodologías de desarrollo de *software*, del modelo de objetos y de sus diagramas arquitectónicos. | Desarrollo de *software*  Modelo de objetos |
| **7** | Jóvenes equipos financieros de empresas de Asia trabajan juntos en una reunión de reflexión sobre el proyecto. Cooperar el trabajo en equipo, la planificación de estrategias, la empresa de inicio de pequeñas empresas o el concepto de coworker de oficinas. Tamaño del cartel  Calendario, reunión o cita de equipo, planificación de proyectos o recordatorio de tiempo, gestión del concepto de cronograma, empresario y empresario que trabaja en la planificación del calendario para programar y organizar el trabajo.  Concepto anual de presupuesto y planificación financiera con el gerente o director ejecutivo de operaciones de diseño o validación de la previsión de ingresos y gastos de la empresa. Finanzas corporativas y plan estratégico. |  | Con la comprensión de estas teorías se podrán entender e interpretar correctamente los ejemplos. Las actividades serán un derrotero para gestionar un proyecto de *software*, pero sea cual sea la metodología que se seleccione, se debe realizar siempre un proceso dedicado y consciente. Esto incluye prácticas para planificar el proyecto, decidir el tamaño y calidad del producto y estimar el costo y la gestión de recursos como el personal y sus tareas concretas. | Planificación del proyecto  Tamaño y calidad del producto  Costo y gestión de recursos |
| **8** | 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  El vector infográfico de modelos de cascada se utiliza en procesos de ingeniería de software o desarrollo de software. La ilustración tiene 6 pasos como la metodología Agile o el pensamiento de diseño para el sistema de aplicaciones  Metodología Ágile vs Cascada para el diagrama de ciclo de vida de desarrollo de software |  | Por otra parte, se busca adquirir conocimientos claros para tener la destreza de seleccionar la metodología adecuada según el proyecto a desarrollar y poder diseñar los elementos estructurales arquitectónicos para un buen desarrollo y buenas prácticas. Para concluir, se desarrollarán temas como metodologías de desarrollo de *software* tradicional y ágil con sus tipos, características, ventajas y desventajas; elementos del modelo, objetos y elementos de estructura arquitectónica con sus tipos y características. | Seleccionar la metodología adecuada |
| **Nombre del archivo** | 228125\_v1 | | | |

**Desarrollo de contenido**

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

|  |
| --- |
| Cuadro de texto |
| Antes de hablar de metodologías de desarrollo de *software* tradicionales conozca qué es metodología. Una metodología es un marco de trabajo que se puede aplicar como guía para el desarrollo de las actividades que se van a llevar a cabo. Por tanto, una metodología de desarrollo de *software* especifica las tareas que se deben seguir, los artefactos que se tienen que construir y las relaciones de los objetos.tareas en el tablero ágil de depuración de software, concepto de metodologías de desarrollo de software ágil, cierre  Desarrollar un *software* de calidad depende de un proceso que contiene actividades y etapas donde la elección de la metodología a seguir genera un impacto trascendental en el éxito del proyecto.  Las metodologías de *software* tradicionales son aquellas que siguen un proceso lineal donde cada una de las etapas del proceso debe ejecutarse secuencialmente, seguida una de la otra. Estas metodologías se caracterizan por ser rígidas; al tener que definir todos los requerimientos desde el inicio del proyecto, no son flexibles ni permiten los cambios. |

**Tipos y características**

|  |
| --- |
| Cuadro de texto |
| Los tipos de metodologías de *software* tradicionales son: |

**Cascada**

|  |
| --- |
| Cuadro de texto |
| Esta metodología de *software* divide el proceso de desarrollo en etapas que se organizan y recorren de arriba a abajo en estricto orden; de ahí se deriva su nombre.  El proceso incluye una serie de etapas, como se presenta en la siguiente figura. |

**Figura 1**

Modelo de cascada

Definición del *software*

Análisis de requerimientos

Diseño de arquitectura

Codificación

Pruebas

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Rutas / Pasos. Verticales 2 | |
| **Introducción** | Amplíe la información sobre cada una de las fases o etapas. | |
| **1** | **Definición del *software***  En esta fase se define el producto desde una visión a alto nivel con los diferentes aspectos importantes que se deben tener en cuenta. | Aprendiendo el concepto de sistema de clasificación LCS con iconos. Aprendizaje automático, minería de datos, modelación de comportamiento, clasificación, inteligencia artificial, reglas, software, iconos de algoritmo. infografía vectorial web  Imagen: 228125\_i3 |
| **2** | **Análisis de requerimientos**  Esta fase comprende las tareas de entender el negocio en su totalidad, de conocer el producto que se quiere desarrollar, sus funciones, sus características, quiénes lo usarán y las relaciones con los sistemas externos. | 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  Imagen: 228125\_i4 |
| **3** | **Diseño de la arquitectura**  Se realiza el diseño de la arquitectura de la solución que se implementará. | Icono de concepto de arquitectura del sistema. Estructura organizativa,ilustración de línea delgada idea modelo. Administración. Jerarquía empresarial, departamentos. Plan estratégico. Dibujo aislado del vector. Trazo editable  Imagen: 228125\_i5 |
| **4** | **Codificación**  En esta etapa se ejecuta el código de la aplicación de acuerdo a lo estipulado en las fases anteriores para cumplir todo lo pactado. | El equipo está desarrollando una aplicación móvil. Un conjunto de ilustraciones vectoriales.  Imagen: 228125\_i6 |
| **5** | **Pruebas**  Fase donde se asegura la calidad del producto de *software*, se revisa que se satisfagan los requerimientos y se reduzcan los riesgos en la implementación en el entorno real. | El equipo está desarrollando una aplicación móvil. Un conjunto de ilustraciones vectoriales.  Imagen: 228125\_i7 |

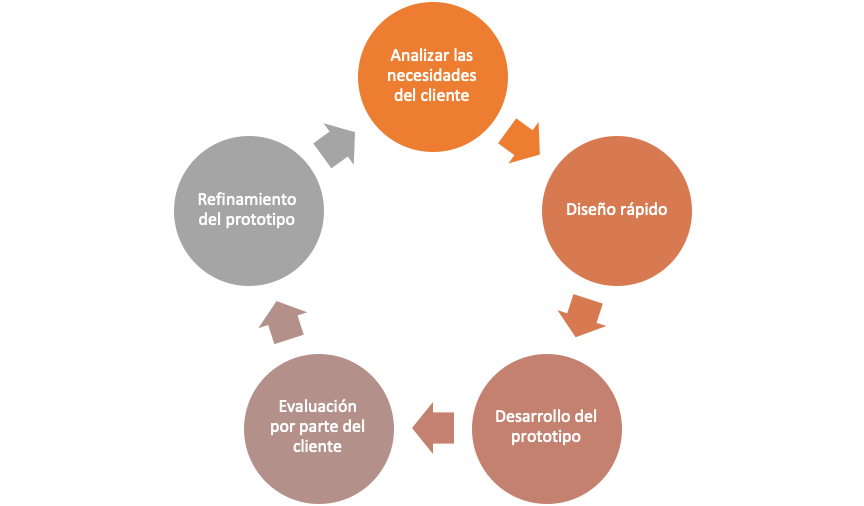
|  |
| --- |
| Cuadro de texto |
| Para poder aplicar esta metodología es necesario que los requerimientos del cliente sean conocidos de antemano o se tengan claros, también se recomienda que sea para un número reducido de requerimientos, teniendo presente que las pruebas se hacen cuando se encuentre desarrollado todo el producto *software*.  Una vez plasmados los requisitos quedan en un estado congelados, lo que dificulta que se cambien o que se incorpore otro requisito al producto; un ejemplo de esto sería que al final de todo el desarrollo, cuando se está probando la calidad del producto en la fase de pruebas, el cliente solicite un nuevo requerimiento por un cambio en su negocio o simplemente porque se le había olvidado. La magnitud de este impacto en el proyecto genera caos porque la fase de requisitos es prácticamente la primera, lo que hace imposible satisfacer esta nueva necesidad.  Las fases no se pueden solapar, solo se inicia una fase cuando la etapa o fase anterior esté totalmente finalizada y probada. Tampoco son aconsejables el reproceso o la revisión de fases anteriores. |

**Prototipado**

|  |
| --- |
| Cuadro de texto |
| El prototipado es una metodología de desarrollo iterativa en la cual se desarrolla una maqueta de la solución. Este prototipo se produce con el fin de mostrarle al cliente si se entendieron los requerimientos iniciales a alto nivel que se hicieron, y si van por buen camino, es decir, se hace el refinamiento junto al cliente.  En este proceso, el cliente estará probando cada uno de los prototipos que se desarrollen desde el equipo de proyecto y emitirá su juicio valorativo, de acuerdo con él, el equipo desecha el prototipo la gran mayoría de veces, o lo termina para irlo evolucionando.  El objetivo principal en esta metodología es poder disminuir el riesgo asociado a los requerimientos y su falta de comprensión.  Gráficamente, el modelo se presenta así: |

**Figura 2**

Modelo de prototipado



|  |
| --- |
| Cuadro de texto |
| Una de las características que tiene esta metodología, la cual puede ser desfavorable, es que al presentarle un prototipo funcional al cliente, este tenga la idea errada de que ya está listo el *software* y solo falten detalles menores.  El usuario / cliente al estar involucrado en el proceso de desarrollo aumenta la aceptación del producto final que será implementado.  Es importante en esta metodología un entendimiento esencial del problema del negocio para evitar generar prototipos y así solucionar problemas incorrectos que provocan gasto de esfuerzo y tiempo de proyecto en prototipos sin fundamento. |

**Espiral**

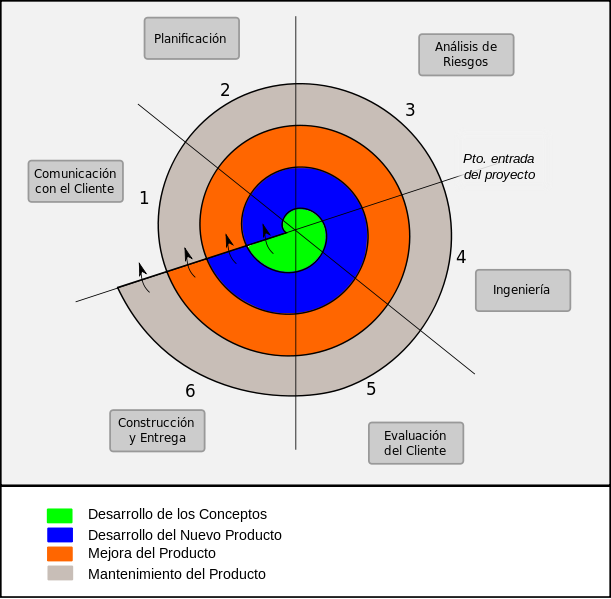
|  |
| --- |
| Cuadro de texto |
| Esta metodología tiene la combinación de los modelos iterativos en su esencia, como el prototipado y el cascada, por lo tanto, es una metodología de carácter incremental.  El modelo cascada ofrece un proceso ordenado estricta y estructuralmente para el proyecto, pero las demandas de la industria de tiempos acelerados o reducidos hace que sus fases sean inapropiadas. Este modelo se basa en ese orden de la cascada volviéndolo iterativo o incremental, teniendo presente que, al ser evolutivo, se aprovechan mejor las funcionalidades del equipo. |

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| En este modelo se divide el *software* en mini proyectos que evoluciona cada uno de acuerdo a las fases que sigue el espiral, comenzando con la comunicación con el cliente para determinar los requisitos iniciales; luego se pasa a la planificación de esos requerimientos donde se incluye la estimación del coste en tiempo y recursos, continuando con el análisis de riesgo, luego empieza la ingeniería del proyecto donde se diseñan las posibles alternativas de prototipo, se presentan al cliente para su evaluación, después se construye, se entrega y se hace la retroalimentación; siempre de la mano del cliente. Si es aceptado, o se va por buen camino, se sigue evolucionando; de lo contrario, se descarta y se comienza de cero nuevamente. | |

|  |
| --- |
| Cuadro de texto |
| La siguiente imagen presenta una explicación gráfica de este proceso. |

**Figura 3**

Modelo espiral



Nota: tomado y adaptado de Deloitte (2020).

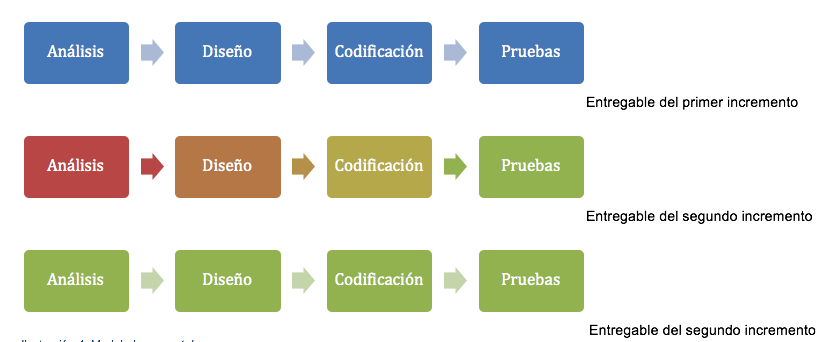
|  |
| --- |
| Cuadro de texto |
| La metodología o modelo en espiral es un enfoque de desarrollo de *software* que surgió como respuesta a los inconvenientes presentados por el modelo cascada. El espiral basa todas sus fases, como se ve en el gráfico, a través de espirales que se van repitiendo hasta entregar un producto final.  Este modelo es global y se usa junto con otros modelos de desarrollo de *software* tanto tradicionales como ágiles.  La característica principal es el control de riesgos que se puede llevar en este modelo porque, como es evolutivo, a medida que se avanza los riesgos se reducen.  Permite una estimación fácil debido a que se hace con mini proyectos donde los requisitos son muy pocos. También, como se evoluciona, hay espacio para atender las retroalimentaciones del cliente. |

**Incremental**

|  |
| --- |
| Cuadro de texto |
| Esta metodología es otro modelo que tiene en cuenta la evolución del producto en el tiempo, iterando la funcionalidad del proyecto. La base fundamental de este modelo es el cascada que se irá iterando repetidamente e incrementando en cada iteración la funcionalidad del producto hasta completar así todos los requerimientos del proyecto.  Es bueno recalcar que de esta metodología, en cada iteración, se obtiene un producto operacional y no un prototipo como en el modelo de prototipado. El siguiente gráfico presenta el esquema del modelo. |

**Figura 4**

Modelo incremental



|  |
| --- |
| Cuadro de texto |
| Una de las características de este modelo de desarrollo es que el *software* se puede empezar a utilizar incluso antes de que se complete totalmente y, en general, es mucho más flexible que las demás metodologías.  Es importante tener claro que cada iteración debe dar como resultado un producto funcional; encargarse de implementar esto puede requerir de un trabajo extra en cada iteración, lo cual es un tiempo y un costo representativos y adicionales al proyecto. |

**RDA**

|  |
| --- |
| Cuadro de texto |
| Esta metodología, llamada así por sus siglas (RDA – Desarrollo Rápido de Aplicaciones) y también RAD por su origen en inglés (*Rapid Application Development*), consiste en tener un desarrollo de *software* lineal secuencial como se estudió en el modelo en cascada, pero llevándolo a un proceso en paralelo con distintos equipos de trabajo.  Estos equipos serán responsables ―cada uno― de una parte de los requerimientos del proyecto. El objetivo de esta metodología es tener en un periodo corto el producto final completo.  Este modelo se caracteriza por sus equipos formados por más o menos seis personas, dentro de las cuales se incluyen desarrolladores y usuarios del sistema de tiempo completo, así como las demás personas involucradas de acuerdo con los requisitos. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Rutas / Pasos Verticales 1 |
| **Introducción** | El esquema de fases o etapas para esta metodología es el siguiente: |
| Modelado de gestión  Modelado de datos  Modelado de procesos  Generación de aplicaciones  Pruebas y entrega  **Imagen:** 228125\_i11 | |
| **Botón 1** | **Modelado de gestión**  En esta etapa se modela la información que gestiona el negocio, la información que el propio negocio genera y el responsable de esta, a dónde se lleva y el responsable de su procesamiento. |
| **Botón 2** | **Modelado de datos**  En este proceso de modelado se deben tener presente los objetos de datos que el *software* va a manejar, cuáles son los atributos de esos datos, cómo se componen y se relacionan entre sí y cuáles procesos de negocio impactan. Lo que se busca como objetivo final es modelar una abstracción muy clara del dominio total de los datos relevantes al problema o producto en cuestión. |
| **Botón 3** | **Modelado de proceso**  Lo que busca este modelado es diseñar el proceso de manipulación de los datos del paso anterior, cómo se agregan, eliminan, modifican o se recuperan. |
| **Botón 4** | **Generación de aplicaciones**  En esta etapa es donde se desarrolla el código, de acuerdo con el lenguaje de programación seleccionado previamente en los requerimientos, y se tratan de reutilizar, siempre que se pueda, los componentes existentes de los procesos anteriores. |
| **Botón 5** | **Prueba y entrega**  Fase donde se realizan las pruebas de lo desarrollado; sin embargo, si se reutilizan componentes, se supone que ya fueron probados en los procesos anteriores y solo falta probar la integración con los nuevos componentes. |

|  |
| --- |
| Cuadro de texto |
| **Tenga en cuenta: l**as etapas anteriores se pueden modificar o variar según los libros, el autor o la página web consultada. |

|  |
| --- |
| Cuadro de texto |
| La característica más importante es que se necesita involucrar al cliente. Esta cooperación es necesaria para resolver dudas y poder avanzar.  También se requiere un gran volumen de recursos para poder realizar el trabajo en varios equipos de manera simultánea, además, se requiere que el *software* a desarrollar pueda ser modularizado. |

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

|  |
| --- |
| Cuadro de texto |
| Las metodologías de desarrollo de *software* ágiles tomaron mayor auge en el año 2001, pero se venían revisando y concretando desde los años noventa. En este nuevo siglo tuvo lugar una reunión de representantes de los modelos ágiles que dio origen a lo que hoy en día se conoce como el manifiesto ágil, donde se encuentra plasmado el espíritu de estas metodologías.  Una parte del manifiesto ágil dice: |

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| “Estamos descubriendo formas mejores de desarrollar *software* tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:  Individuos e interacciones sobre procesos y herramientas.  *Software* funcionando sobre documentación extensiva.  Colaboración con el cliente sobre negociación contractual.  Respuesta ante el cambio sobre seguir en un plan.  Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda”. (Manifiesto agile, s.f.) | |

|  |
| --- |
| Cuadro de texto |
| Como se pudo ver en el tema anterior, las metodologías tradicionales son estrictas y rígidas, en cambio, las ágiles buscan la flexibilidad en el proceso tratando de llegar a un equilibrio entre restricciones y el libre desarrollo.  Estas metodologías requieren una comprensión y aceptación del cliente; donde su papel cambia, se necesita una relación más estrecha. En este tipo de enfoques no existe el contrato fijo, es decir, no se fijan desde el inicio costo, tiempo, recursos totales ni alcance. El alcance debe variar de forma controlada para que durante el proceso el cliente defina sus requerimientos. |

|  |
| --- |
| Cuadro de texto |
| Existe una gran variedad de metodologías ágiles, las cuales son evolutivas, iterativas y tienen tiempos fijos en sus iteraciones.  A continuación conozca cada una de ellas: |

**Scrum**

|  |
| --- |
| Cuadro de texto |
| Esta metodología ágil está muy empoderada en la industria, su nombre viene del deporte *rugby*, donde todos los miembros del equipo deben hacer estrategias, unir fuerzas y empujar sus obstáculos para llegar al éxito que, en el caso del juego, serían los jugadores del otro equipo. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Infografía interactiva Punto caliente | |
| **Texto introductorio** | Esta metodología tiene una serie de componentes que se pueden agrupar en las categorías roles, activos y flujo de trabajo. | |
| Realizar una imagen como la que se indica. Los textos editables se encuentran en la carpeta Anexos con el nombre Infografia\_interactiva.pptx | | |
| **Código de la imagen** | 228125\_i12 | |
| **Punto caliente 1** | Es una persona definida por el cliente, tiene que conocer muy bien el negocio y estar en todo el proceso del proyecto como un involucrado encargado de definir el *software*. | *Product Owner* |
| **Punto caliente 2** | Es el rol encargado de quitar todos los posibles obstáculos del proyecto y engranar todas las piezas para que funcionen bien, en pocas palabras, es un facilitador, y junto con el *Product Owner*, se ocupa de definir la operatividad del *software*. | *Scrum Master* |
| **Punto caliente 3** | Equipo de desarrolladores conformado por entre 3 y 7 personas. | *Team* |
| **Punto caliente 4** | Integra el proyecto a la organización y es el encargado de los recursos y su aprobación. | *Project Manager* |
| **Punto caliente 5** | Es un listado de requerimientos ya priorizados del sistema. | *Product Backlog* |
| **Punto caliente 6** | Es el conjunto que se selecciona de requerimientos provenientes del *Product Backlog* y será desarrollado por el equipo en una iteración llamada *Sprint*. | *Sprint Backlog* |
| **Punto caliente 7** | Modelo o diagrama que muestra el trabajo sin terminar. Se puede hacer para el *Sprint* como también para el proyecto completo. | *Burdown Char* |
| **Punto caliente 8** | Proceso donde se planifica todo el proyecto. | *Project Planning* |
| **Punto caliente 9** | Tarea donde se lleva a cabo la planificación de la iteración. | *Sprint Planning* |
| **Punto caliente 10** | Es una iteración de duración fija y dura entre 2 y 4 semanas. | *Sprint* |
| **Punto caliente 11** | Versión del código funcional entregado al cliente. | *Release* |
| **Punto caliente 12** | Reunión diaria que se lleva a cabo entre el *Scrum Master* y el equipo para revisar las tareas del día. | *Daily Scrum* |
| **Punto caliente 13** | Reunión que se realiza al finalizar el *Sprint* con el objetivo de mostrarle al cliente la nueva funcionalidad. | *Demo Meeting* |
| **Punto caliente 14** | Como su nombre lo indica, el equipo con el *Scrum Master* revisan el *Sprint* que acaban de terminar y establecen cuáles son las fallas que se tuvieron y cómo mejorar para los próximos. | *Sprint Retrospective* |

|  |
| --- |
| Cuadro de texto |
| La anterior imagen muestra todo lo que maneja la metodología SCRUM y si es la adecuada para la implementación en el proyecto. |

**Kanban**

|  |
| --- |
| Cuadro de texto |
| Esta metodología tiene sus orígenes en los procesos de producción ideados por Toyota, donde se empezó con el uso de tarjetas para tener clara la cantidad de material que necesitaban en cada proceso de la cadena de producción.Gestor que trabaja con el desarrollo ágil de productos utilizando una metodología de limpieza o kanban, un marco de gestión de proyectos magro con una estrategia iterativa o incremental, una persona con un equipo en la oficina  Kanban significa en japonés tarjetas visuales; la palabra *kan* es visual y la palabra *ban* es tarjeta. Hace parte de las metodologías ágiles cuyo objetivo es gestionar de forma óptima cómo se van completando las tareas y qué recursos tienen. Es así como Kanban es una metodología que busca apoyar a los equipos a encontrar un balance entre las tareas pendientes por realizar y la disponibilidad de cada integrante, su filosofía es la mejora continua a través de la técnica de extracción, es decir, se tiene un repositorio o lista de tareas de la cual se van tomando y se van anexando al flujo de trabajo apoyados en los tableros Kanban. |

**Figura 5**

Tablero Kanban



|  |
| --- |
| Cuadro de texto |
| La anterior gráfica es el tablero Kanban, el cual se utiliza para organizar el trabajo, cada tarjeta es una tarea y va avanzando en el tablero según su desarrollo. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Rutas / Pasos. Verticales 1 |
| **Introducción** | Kanban tiene cuatro principios que son fundamentales para su implementación: |
| Ilustración vectorial de las grandes tarjetas de transferencia de equipo informático y programador en el tablero de tareas kanban electrónico. Aplicación kanban en línea, método ágil de gestión de proyectos visuales, conceptos de trabajo en equipo.  **Imagen:** 228125\_i15 | |
| **Botón 1** | **Empiece con lo que hace ahora**  Esta metodología puede ser implementada en cualquier proceso de trabajo porque es lo suficientemente flexible para adaptarse. |
| **Botón 2** | **Comprométase a buscar e implementar cambios progresivos y evolutivos**  Los cambios muy grandes pueden impactar negativamente el equipo, por la demora, el esfuerzo y los posibles arreglos que toque realizar; por eso se deben hacer pequeños cambios e ir evolucionando. |
| **Botón 3** | **Respete los roles, los procesos y las responsabilidades actuales**  Kanban no tiene roles integrados o definidos, puede integrarse y funcionar con la jerarquía de su equipo. |
| **Botón 4** | **Impulse el liderazgo en todos los niveles**  Se habla de un liderazgo horizontal, y no solo vertical, dentro del proyecto o equipo de trabajo. |

|  |
| --- |
| Cuadro de texto |
| Los principios dan una guía para el equipo al momento de comenzar el flujo de trabajo, pero se deben seguir las siguientes prácticas que usan grandes empresas para lograr una mejora continua y un crecimiento progresivo:   * **Visualizar el trabajo**: Kanban es visualmente muy claro. Una tarjeta de trabajo viajará del lado izquierdo al lado derecho del tablero comenzando por cada una de las columnas que representan un estado, hasta que llegue a estar finalizada. * **Limitar el trabajo en curso**: Kanban se basa en un principio de movimiento rápido, es decir, que cada tarjeta o tarea avanza rápido en el tablero y no se queda mucho tiempo en “trabajo en progreso”; no hay una regla, pero se debe poner un indicador limitante para saber cuántas tareas al tiempo pueden estar en progreso. * **Gestionar el flujo de trabajo**: mejorar la gestión del flujo de trabajo ayudará a que se tenga control sobre el avance de las tareas y así se logre cumplir el principio anterior. * **Implementar políticas explícitas de procesos**: las políticas de las tareas y su sintaxis o reglas deben estar claras. Todo el equipo debe conocerlas. * **Implementar ciclos de comentarios**: se deben gestionar muy bien los comentarios, recopilarlos de ambas fuentes: del cliente y del equipo de desarrollo. Sean dudas, preguntas, soluciones y demás comentarios pertinentes. * **Mejorar colaborando y crecer experimentando**: indica que se pueden tener otras metodologías fomentando el trabajo junto con Kanban. Esta colaboración mejora los procesos.   Los anteriores son los conceptos y la información fundamental para saber si el equipo necesita esta metodología y si puede empezar a implementarla. |

**Lean**

|  |
| --- |
| Cuadro de texto |
| Esta metodología busca la mejora continua y la eliminación del desperdicio. El desperdicio se refiere a todo lo que no aporta o suma al desarrollo del proyecto y, en consecuencia, al cliente.  Lo anterior no se trata de eliminar, como lo indica la palabra, sino de generar valor durante el flujo de trabajo, optimizando así el esfuerzo realizado en cada tarea, para lo cual es necesario siempre estar de la mano del cliente, y escucharlo.  Los fracasos se toman como aprendizajes; del error se aprende para no volverlo a cometer. Cuando una organización implementa Lean se caracteriza por ser metódica y buscar siempre el diferenciador con su competencia.  Fomenta la mentalidad de ensayo y error bajo su pilar de una gestión de riesgo e innovación eficiente.  Se debe respetar el equipo de trabajo sin importar su jerarquía; las ideas innovadoras siempre provienen de las personas que están directamente involucradas con el proceso de fabricación del producto. Este respeto fomenta un liderazgo horizontal y no vertical como las compañías comunes, y ese liderazgo Lean empodera a todos los empleados para la toma de decisiones, el aporte de nuevas ideas al proceso con el objetivo de ejecutar el trabajo siempre de la mejor forma y responder así ante cualquier situación. |

**Programación Extrema XP**

|  |
| --- |
| Cuadro de texto |
| Por sus siglas en ingles XP (*Extreme Programming*) fue desarrollada por Kent Beck y busca la disciplina extrema de las personas que conforman el equipo de desarrollo aplicando unas prácticas consideradas esenciales en el proceso. Se fundamenta en valores, principios y prácticas. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Carrusel de tarjetas | |
| **Introducción** | Los valores incluyen los de todo el grupo que conforma el proyecto, en este caso se refiere a la forma como toman las decisiones frente a una situación específica; son cinco y se describen a continuación: | |
| **Comunicación**  Es el valor clave en el proceso de desarrollo de *software* y más específicamente en cada equipo de trabajo; junto con este valor se promueven la colaboración y el aprendizaje en equipo. | | personas de diálogo,icono de personas que hablan,vector e ilustración  **Imagen:** 228125\_i16 |
| **Simplicidad**  Debido a varias lecciones aprendidas en los proyectos, se pone la simplicidad como base para el diseño y la codificación del *software*. Esto indica que solo se debe hacer lo que se pida, nada más. | | Reparar iconos - Iconos de línea de vectores. Stroke editable. Gráfico vectorial  **Imagen:** 228125\_i17 |
| **Retroalimentación**  Este valor indica aprender sobre lo ya realizado, es decir, sobre lo que ya se probó y se culminó. Aquí nace una práctica clave llamada refactorización. | | Conjunto de iconos de línea web de negocios y finanzas. Testimonios, gestión de relaciones con el cliente o concepto de CRM. Colección de símbolos de estilo de contorno simple. Ilustración vectorial aislada en fondo blanco. EPS 10.  **Imagen:** 228125\_i18 |
| **Coraje**  Como su nombre lo indica, este valor crea e impulsa la actitud positiva frente a cualquier situación. Siempre se dice la verdad ante cualquier pregunta sobre el proyecto. | | diseño vectorial de iconos de amistad negro  **Imagen:** 228125\_i19 |
| **Respeto**  Para tener un grupo compenetrado la base de todo es el respeto entre las personas y sus interacciones. | | Icono de apretón de manos. Símbolo del trato. Estilo de línea  **Imagen:** 228125\_i20 |

|  |
| --- |
| Cuadro de texto |
| Los valores que se acabaron de revisar no indican cómo se deben llevar a cabo las tareas del proyecto, son las prácticas las que muestran cómo se deben ejecutar en detalle. Pero como hay un espacio muy grande entre los valores y las prácticas, XP dispuso de unos principios que permiten la generación de prácticas concretas a partir de los valores.  En XP existen 15 principios que se describirán brevemente a continuación:   * **Beneficio mutuo**: todas las tareas o actividades realizadas deben beneficiar a todos los miembros del equipo. * **Diversidad**: este principio indica que se debe ser diverso en el proceso de desarrollo de proyecto; todas las opiniones cuentan. * **Humanidad**: se deben valorar las necesidades humanas en el ciclo de vida del proyecto. * **Responsabilidad aceptada**: la responsabilidad no puede ser impuesta, debe aceptarse. * **Economía**: todas las actividades que se realicen deben aportar valor al negocio. * **Calidad**: el equipo debe trabajar con calidad y sentirse orgulloso de cada producto que desarrolle. * **Mejora**: se debe velar siempre por una mejora continua en los procesos. * **Flujo**: hay que asegurar el transcurrir normal de los flujos y la entrega de productos. * **Reflexión**: siempre se debe tener un proceso de reflexión ante cada trabajo realizado. Cómo y por qué. * **Pequeños pasos**: siempre se debe avanzar de manera constante y con pequeños pasos. * **Autosemejanza**: aprender de las experiencias pasadas y lecciones aprendidas. * **Falla**: no quebrarse ante una falla, sino utilizar ese caso como parte del aprendizaje. * **Oportunidad**: los problemas que se presenten en el desarrollo del proyecto hay que transformarlos en oportunidades de mejora. * **Redundancia**: tomar los problemas, buscar las diferentes soluciones y probar varias al tiempo. |

|  |
| --- |
| Cuadro de texto |
| XP, aparte de ofrecer valores y principios, tiene prácticas que proponen una mejor organización, planificación, comunicación y permiten generar una cultura de excelencia. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | | Pestañas o *vertical tabs* |
| **Introducción** | | Las prácticas más relevantes son: |
| Extreme Programming - XP | Cátedra Viewnext USAL  **Imagen:** 228125\_i21 | | |
| **El juego de la planificación** | Esta práctica es muy importante y requiere una participación activa del cliente en la planificación de los requerimientos junto con el equipo en busca de la cohesión entre el conocimiento del negocio y los requerimientos por parte del cliente, y el de costos y desarrollo del equipo. | |
| **Pequeños *releases*** | El equipo de desarrollo debe velar por entregar con frecuencia al cliente versiones de las iteraciones funcionales. | |
| **Metáforas** | Capacidad de explicar el sistema a personas nuevas con base en el concepto de metáforas con el fin de evitar el tiempo que gastarían leyendo la documentación. | |
| **Diseño simple** | La idea es hacer solo lo que se solicita y mantener todo con simplicidad. | |
| **Desarrollo guiado por pruebas** | Es un enfoque evolutivo que permite el desarrollo de actividades con calidad. Para ello se crea primero la prueba con la cual se va a testear y después se hace la mejora. | |
| **Refactorización** | Consiste en mejorar el código ya realizado sin alterar su comportamiento. | |
| **Programación en parejas** | Es una técnica de trabajo que reúne a un equipo de dos personas en un solo computador para que ambas trabajen resolviendo el mismo problema. | |
| **Propiedad colectiva del código** | Esta práctica indica que el código no se privatiza o ciertas partes no son de equis desarrollador; cualquier desarrollador del equipo puede desarrollar o mejorar cualquier funcionalidad. | |
| **Integración continua** | Indica que se requiere que todo se vaya integrando y probando de forma continua, sin esperar hasta el final para unir todas las partes. | |
| **Historias de usuario** | Es un documento importante donde se describen las funcionalidades del proyecto. | |
| **Estándares de codificación** | Consiste en producir código con un estilo estandarizado para todo el proyecto y no que cada desarrollador lo haga a su estilo. | |

**RUP**

|  |
| --- |
| Cuadro de texto |
| El Proceso Unificado de Rational o Rational Unified Process (RUP) es una metodología que combina buenas prácticas de gestión de proyectos ágiles y tradicionales. La política de RUP es desarrollar el proyecto en iteraciones que van evolucionando gradualmente. Cada iteración se compone de cuatro fases que son su ciclo de vida y son guiadas por casos de estudio. El inicio se da con requerimientos genéricos y se va avanzando hasta finalizar en los más especializados. Metodología de desarrollo de software RUP de proceso unificado racional, esquema de proceso de marco detallado. Administración de proyectos, ciclo de vida del flujo de trabajo de productos. Inicio, elaboración, construcción y transición.  Existe otro método llamado AUP (proceso unificado ágil), el cual es una versión simplificada de RUP. Se basa en el desarrollo de aplicaciones empresariales y en él se usan técnicas ágiles llamadas TDD (desarrollo basado en pruebas), MDD (desarrollo basado en modelos) y gestión de *software*. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | | Pestañas o *vertical tabs* |
| **Introducción** | | Las cuatro fases en las que se divide esta metodología son: |
| Three rotating gears. Unified process. Adapting to change. Vector illustration isolated on white background.  **Imagen:** 228125\_i23 | | |
| ***Launch*** | Es el inicio del proyecto donde se identifica el alcance, la arquitectura a alto nivel, se vinculan todos los interesados del proyecto y se elabora el presupuesto. | |
| ***Design*** | Se diseña todo el *software*, incluyendo la arquitectura detallada. | |
| ***Realization*** | Proceso de codificación del *software* que se hace de forma gradual con priorización de los requerimientos. | |
| ***Delivery*** | Implementación y pruebas del *software*. | |

|  |  |
| --- | --- |
| **Tipo de recurso** | Tarjetas animadas |
| **Introducción** | Las metodologías ágiles tienen las siguientes características en común: |
| Beneficio del icono de riesgo financiero al reducir la reducción del dólar  **Imagen:** 228125\_i24 | **Menos costes**  Siempre se trata de mejorar la calidad del producto y que se haga solo lo que le dé valor al cliente a fin de maximizar las ganancias. |
| Iconos vectores de línea de personas que hablan  **Imagen:** 228125\_i25 | **Mejores interacciones con el cliente**  Se busca optimizar la interacción con el cliente, por este motivo la comunicación debe ser fluida y en ambas direcciones. Se deben tener unos tiempos eficientes de respuesta. |
| Plantilla de diseño de vectores de iconos de marca de rosette. Stroke editable.  **Imagen:** 228125\_i26 | **Mayor calidad**  Lean indica que se debe optimizar cada uno de los procesos y tareas prestando mucha atención en las especificaciones y sus detalles con el objetivo de reducir los defectos y retrasos. Con esto se optimizan los costos y se impacta en una mejoría en tiempos durante el proyecto. |
| Un gráfico de crecimiento con flechas circulares en el icono de diseño de línea, concepto de mejora continua  **Imagen:** 228125\_i27 | **Crea cultura de mejora**  Así como en una empresa, un equipo de proyecto debe tener cultura: la de la comunicación, por ejemplo; pero en este caso se enmarca en la importancia de la calidad, que se vuelve cultura al crear estrategias, prácticas y formas de trabajo que se llevan a cabo en cada tarea realizada para que el trabajo sea más eficiente. |
| Un lugar de trabajo inclusivo. Ilustración vectorial del icono de contorno relleno de protección del empleado. Color editable. EPS 10  **Imagen:** 228125\_i28 | **Mayor moral de los empleados**  Esto se da gracias al liderazgo horizontal, donde los jefes toman en cuenta las opiniones de los trabajadores y sus ideas se valoran, se revisan, se analizan y, de ser necesario, se implementan. |

|  |
| --- |
| Cuadro de texto |
| Para finalizar, encontramos las siguientes ventajas y desventajas de estas metodologías.  Ventajas:   * Incentiva la creatividad * Entregas continuas * Mejora continua del flujo de trabajo * Flexibilidad ante los cambios * Se eliminan los cambios posteriores a los proyectos ya terminados * Mejora la comunicación del equipo e involucra más estrechamente al cliente.   Desventajas:   * La excesiva revisión puede llevar a retrasos * Sin un equipo de trabajo calificado las entregas pueden ser tardías. * Sin información detallada de los cambios se puede llevar a operaciones improductivas. |

1. **Elementos del modelo objetos**

|  |
| --- |
| Cuadro de texto |
| OOP - Acrónimo de programación orientada a objetos. fondo de concepto de negocio.  concepto de ilustración vectorial con palabras clave e iconos. ilustración con iconos para banner web, volante, aterrizaje  Cuando se habla del modelo de objetos se hace referencia a los modelos del paradigma de programación orientado a objetos; es decir, los conceptos, sintaxis, fundamentos y pilares que lo componen.  El paradigma de programación orientado a objetos es un estándar en la industria del *software* y es implementado en diferentes lenguajes de programación como Java, Net, PHP, C#, entre otros. La programación orientada a objetos se centra en las clases y en los objetos como su *core* de programación. |

**Objeto**

|  |
| --- |
| Cuadro de texto |
| Primero se debe definir qué es un objeto. En diferentes libros hay definiciones de este concepto, pero se dirá lo siguiente: |

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Un objeto es una entidad del mundo real según el contexto - problema que se esté trabajando. Tiene características (atributos) y comportamientos (métodos) específicos. | |

|  |
| --- |
| Cuadro de texto |
| Por ejemplo, si se está trabajando un *software* para una universidad, el universo de dominio del problema trae varios objetos: los docentes, los estudiantes, los administrativos, etc.  Si se analiza, el estudiante tiene características que lo definen; lo que se llama atributos: código, nombre, dirección, número de documento, teléfono, etc.; y tendrá un comportamiento específico dentro del problema, como ver notas, matricularse, materias, etc. |

**Clase**

|  |
| --- |
| Cuadro de texto |
| Después de tener clara la idea de un objeto se debe revisar qué es una clase en este paradigma. La clase es como una plantilla. Si se hace una analogía, es como el plano de una casa, de ese plano se pueden crear muchas casas. En este caso, se podrían crear muchos objetos con ese plano llamado clase.  La clase es una entidad que reúne los objetos con atributos y métodos en común, y de la cual van a crearse los objetos. Un objeto es una instancia de una clase: vive y muere en memoria, en tiempo de ejecución del programa o *software*.  Más adelante se revisará cómo es una clase visualmente en los diagramas UML que se realizan en el diseño del *software*. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Infografía estática |
| **Texto introductorio** | A continuación se muestran los pilares de la programación orientada a objetos: |
| Realizar una infografía como se indica. Los editables se encuentran en la carpeta Anexos con el nombre Infografia\_estatica.pptx | |
| **Código de la imagen** | 228125\_i30 |

1. **Elementos de la estructura arquitectónica**

|  |
| --- |
| Cuadro de texto |
| Los elementos de la estructura arquitectónica son los elementos de diseño del proceso de desarrollo de *software*. Para empezar se debe indicar que un artefacto es un producto que resulta del mismo proceso de desarrollo y puede ser un documento, un modelo o un elemento en particular. |

**Diagrama de base de datos**

|  |
| --- |
| Cuadro de texto |
| El diagrama de base de datos se refiere al modelo entidad relación, también llamado modelo conceptual de datos. Fue desarrollado por Peter Chen en 1976 y consiste en una práctica dinámica que se caracteriza por una representación gráfica que incorpora información relativa a los datos y sus respectivas relaciones, usado como una vista general de los datos a implementar y la estructura en la organización.  El modelo entidad / relación sirve como guía para el paso siguiente que es la construcción de la base de datos en un sistema gestor de bases de datos. |

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 1 |
| **Introducción** | Los elementos de un modelo de entidad / relación son: |
| Heli Sulbaran: Diagrama Entidad-Relación  **Imagen:** 228125\_i31 | |
| **Entidades**  Las entidades se representan gráficamente mediante un rectángulo que en el interior lleva el nombre correspondiente. El nombre de la entidad no se debe repetir, cada entidad lleva su nombre.  Ejemplos de entidades: profesores, empleados, estudiantes, clientes, proveedores, etc. | |
| **Relaciones**  Las relaciones son las asociaciones que se describen entre dos atributos (campos) de dos tablas para compartir información. Esas respectivas relaciones entre las entidades se representan por medio de un rombo y describiendo en el interior la acción de la relación. | |
| **Atributos**  Una entidad está representada por un grupo de atributos, los cuales son los encargados de describir una entidad específica. Como los atributos se encargan de describir cada una de las características o propiedades de la entidad, se representan por medio de una elipse o un círculo con su nombre correspondiente en el interior.  Por ejemplo, la entidad vehículo se puede describir con los atributos marca, modelo, color, placa, número de pasajeros, etc. | |
| **Cardinalidad**  Esta cardinalidad dentro del modelo se representa en los extremos de cada relación. La cardinalidad expresa cuántas entidades de un extremo de la relación están relacionadas con cuántas entidades del otro extremo. Pueden ser “uno a uno”’, “uno a varios”’ o “varios a varios”. | |
| **Clave en una entidad**  Dentro del modelo entidad / relación están representadas:   * La clave primaria: es aquel atributo único para cada registro. Se representa por medio de una línea continua que subraya el nombre del atributo correspondiente. * La clave foránea: es la encargada de demostrar la respectiva relación que se tiene en dos entidades; se representa marcando una línea subrayada de forma punteada debajo del nombre del atributo - candidato de la relación. | |

**Diagrama de clases**

|  |
| --- |
| Cuadro de texto |
| Antes de iniciar a hablar sobre el diagrama de clases ―que muestra la estructura estática del sistema, sus clases y sus relaciones― se debe revisar qué es una clase y cómo se ve.  Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.  La siguiente imagen presenta un modelo de una clase. |

**Figura 6**

Modelo de una clase



|  |
| --- |
| Cuadro de texto |
| Los diagramas de clases son los más utilizados en el modelado de sistemas orientados a objetos. Es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellas.  Son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, los componentes que se encargarán del funcionamiento y la relación entre uno y otro.  Describe gráficamente las especificaciones de las clases de *software* y de las interfaces (por ejemplo, las de Java) en una aplicación.  Contiene la siguiente información:   * Clases, asociaciones y atributos * Interfaces, con sus operaciones y constantes * Métodos * Información sobre los tipos de los atributos * Navegabilidad * Dependencias   Para elaborar un diagrama de clases orientado al diseño se debe tener en cuenta lo siguiente:   * Identifique todas las clases que participan en la solución del *software* * Dibújelas en un diagrama de clases * Identifique los atributos provenientes de los conceptos asociados del modelo conceptual * Agregue los nombres de los métodos * Incorpore la información sobre los tipos a los atributos y los métodos * Agregue las asociaciones necesarias para dar soporte a la visibilidad requerida de los atributos * Agregue flechas de navegabilidad a las asociaciones para indicar la dirección de la visibilidad de los atributos. |

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | | Pestañas o *vertical tabs* |
| **Introducción** | | Los datos que componen la clase son: |
| **Imagen:** 228125\_i33 | | |
| **Atributo** | Son valores que corresponden a un objeto; como color, material, cantidad, ubicación. Generalmente se conoce como la información detallada del objeto. Suponiendo que el objeto es una persona, sus propiedades serían: nombre, edad, sexo, etc.  Tipo: puede llegar a depender del lenguaje de programación a utilizar (*public, protected, private*).  Valor inicial: valor que poseerá el atributo al crear un objeto.  Visibilidad: está relacionada con el encapsulamiento.  Multiplicidad: determina si un atributo debe estar o no y si posee un único valor o una lista de valores.  1-> El atributo debe tener un único valor.  0...1-> El atributo puede o no tener un valor.  0...\* -> El atributo puede tener varios valores o ninguno.  1...\*-> El atributo puede tener varios valores, pero debe tener al menos uno.  \* -> El atributo puede tener varios valores.  m…n ->El atributo puede tener entre m y n valores. | |
| **Visibilidad** | El encapsulamiento presenta las siguientes ventajas básicas:   * Se protegen los datos de accesos indebidos. * Favorece la modularidad y el mantenimiento.   Los atributos de una clase no deberían ser manipulables directamente por el resto de los objetos. | |
| **Niveles de encapsulamiento** | (-) Privado: es el más fuerte. Esta parte es totalmente invisible desde fuera de la clase.  (~) *Package*: solo es visible dentro del mismo *package*.  (#) Los atributos / métodos protegidos están visibles para las clases amigas y para las clases derivadas de la original.  (+) Los atributos / métodos públicos son visibles a otras clases. | |
| **Método** | Una operación (método) es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento. Es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase.  Para cada operación debe especificarse:   * Tipo retornado : puede llegar a depender del lenguaje de programación a utilizar. * Parámetros: especificación del tipo de datos y la información que determina el funcionamiento de la clase. * Visibilidad: está relacionada con el encapsulamiento (-, #, +). | |

**Diagrama de casos de uso**

|  |  |
| --- | --- |
| **Tipo de recurso** | Acordeón tipo 2 |
| **Introducción** | El diagrama de casos de uso se utiliza para modelar todas las funcionalidades del sistema. Se basa en casos de uso y utiliza tres elementos básicos: |
| **Imagen:** 228125\_i34 | |
| **Actores**  Representa los diferentes papeles que una persona u otro sistema pueden tener en la interacción con el sistema en construcción. | |
| **Casos de uso**  Es el elemento que representa la funcionalidad que el actor puede realizar con el sistema. | |
| **Relaciones**  Son las formas como se asocian los casos de usos, los actores y casos de uso y actores. | |

|  |
| --- |
| Cuadro de texto |
| Caso de uso  Un caso de uso es una secuencia de acciones que ejecuta un actor que dan un resultado de valor y observable ―incluyendo sus variantes―y que el sistema puede realizar.  Los casos de uso son para modelar las funcionalidades del sistema, el qué hace el actor con el sistema y no el cómo. El nombre de un caso de uso debe comenzar con un verbo para especificar que se trata de un proceso.  Los requerimientos funcionales están naturalmente estructurados como casos de uso.  ¿Cómo identificar casos de uso? Hay que preguntarse:  1. ¿Cuáles son las tareas y responsabilidades de cada actor con el sistema?  2. ¿Algún actor creará, almacenará, cambiará, borrará o leerá información del sistema?  3. ¿Qué casos de uso crearán, almacenarán, cambiarán, borrarán o leerán información?  Actores  Especifica un conjunto de roles que los usuarios de los casos de uso representan al interactuar con estos.  Un actor puede ser el rol que desempeña una persona o un sistema externo que interactúa con nuestro *software*.  Los actores son los que ejecutan los casos de uso.  Para complementar el tema se le invita a ver el siguiente video. |

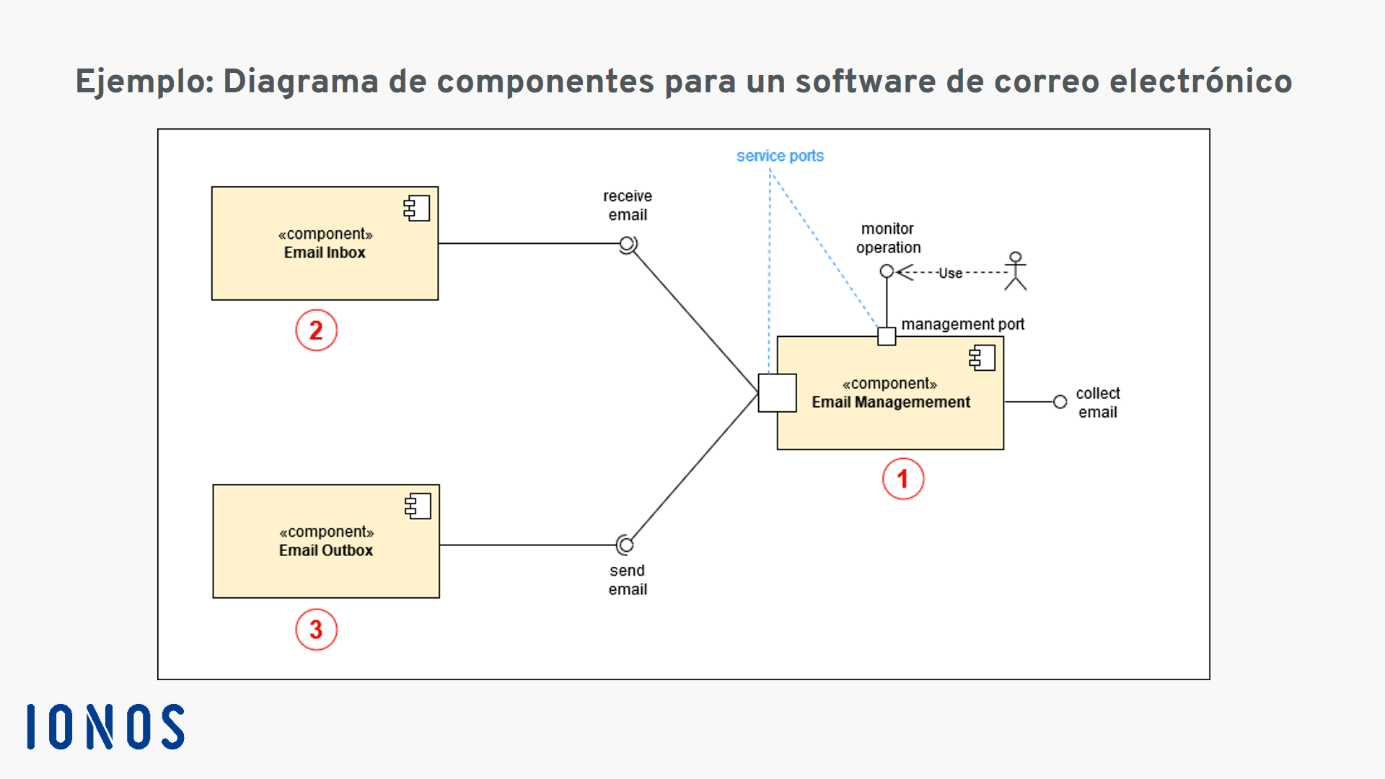
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de recurso** | Video clase / tutorial | | | |
| **NOTA** |  | | | |
| **Título** | Diagrama de casos de uso | | | |
| **Escena** | **Imagen** | **Sonido** | **Narración** | **Texto** |
|  | Nota: el video se encuentra en la carpeta Anexos con el nombre Diagrama de casos de uso.mp4  Se debe grabar el audio y colocarlo en el video.  El video tiene el audio del experto para que se guíe cuando se coloque el audio nuevo. |  | En el componente formativo se pudieron observar algunos diagramas. Es importante decir que todos estos diagramas se hacen a través del lenguaje UML que es el lenguaje unificado de modelado que me permite hacer todo el diseño de *software*.  En este caso nos ubicamos directamente en el diagrama de casos de uso. Este diagrama permite modelar los actores, los casos de uso, y sus relaciones; y esto lleva directamente a al relacionamiento con todo lo que son los requerimientos funcionales. La herramienta para trabajar es UML y, en este caso, se puede utilizar cualquier otra herramienta que permita hacer diseños UML.  Esta es la interfaz que se ve acá.  En la parte derecha está el proyecto definido, no tiene ningún nombre; se pulsa clic derecho y se tiene la opción de agregar diagrama; en este caso, se busca el diagrama que se llame diagrama de casos de uso.  Al momento de agregarlo se observa que deja esta parte central en blanco, es decir, abre un espacio de trabajo. Y ubica una barra de herramientas que tiene todos los componentes posibles de utilizar en mi diagrama de casos de uso; paquetes, la parte de los casos de uso, el actor y las relaciones que se pueden tener.  En este caso, se explica ahora un diagrama de casos de uso sencillo. Se tomará como ejemplo el cajero automático.  Si se toma el sistema del cajero automático, se toma un solo actor; en ese sentido sería el **actor cliente**, el que se acerca al cajero automático a utilizar ese sistema. Se coloca aquí, se da clic, se da clic acá, y un actor; entonces se coloca el actor cliente. Ese actor cliente ¿qué puede hacer?  Recordar que las funciones que ese actor realiza con el sistema se refieren a los casos de uso, porque caso de uso es una secuencia de acciones que dan un resultado observable y de valor al actor. Entonces, un caso de uso sería **retirar dinero**; porque se interactúa con el cajero automático, o con ese sistema, para retirarlo.  Otro caso de uso sería **consultar dinero** porque en el cajero se consulta el dinero de una cuenta; uno más es el que se presenta en los cajeros más recientes, que es el de **consignar**, pues hay unos cajeros en los cuales ya se puede consignar dinero, Se le dice que se va a consignar dinero.  Aparte se dice que existen otros casos de uso posibles de hacer. ¿Cuáles? Aquí se puede decir **transferir dinero**; es posible entrar al cajero electrónico a transferir dinero.  Incluso, ahora hay una opción nueva y es el **retiro de Nequi**. Se entra al cajero electrónico y se retira de Nequi.  Entonces, ¿qué sucede con esto? ¿Qué relación existe entre el actor y los casos de uso? Siempre es una **relación de asociación** porque la asociación quiere decir **uso,** y yo, como actor, uso, ejecuto el caso de uso; ubico acá, doy clic en el centro y desplazo hasta el centro del caso de uso; ahí me quedo, doy clic en el centro, desplazo hacia el caso de uso, doy clic en el centro, desplazo hacia el caso de uso, doy clic en el centro, desplazo hacia el caso de uso, doy clic y desplazo hacia el caso de uso y ahí me quedaría la relación del actor; en ese caso, el actor - cliente con los casos de uso.  En este caso se representa el sistema de cajero electrónico, y se tiene una primera versión del diagrama de casos de uso. ¿Por qué una primera versión? Porque es el primer acercamiento que se hace y se debe ir puliendo porque existen más relaciones, existen más detalles; por ejemplo, la relación de extensión, de inclusión, de generalización. Y pueden existir más actores: un actor como el banco, que es un actor externo, pero es un actor que también interactúa con el sistema; entonces, se puede expandir un poco más este diagrama de casos de uso, de acuerdo con el contexto de ejemplo que se da y que se puede revisar. Aquí está el acercamiento, el diagrama de casos de uso del sistema cajero electrónico. |  |
| **Nombre del archivo** | 228125\_v2 | | | |

**Diagrama de componentes**

|  |
| --- |
| Cuadro de texto |
| El diagrama de componentes modela los artefactos físicos o lógicos del sistema, en UML representa los módulos.  Los componentes encapsulan clases del sistema y por eso se les conoce como subclases, tienen estructura compuesta y define los mismos contenidos de la clase.  La siguiente imagen presenta un ejemplo de un diagrama de componentes de correo electrónico. |

**Figura 7**

Ejemplo diagrama de componentes de correo electrónico



Nota. Tomado y adaptado de IONOS (2020).

**API**

|  |
| --- |
| Cuadro de texto |
| Las API son componentes que permiten que dos *software* se comuniquen entre sí a través de un protocolo de comunicación; tiene diferentes definiciones y restricciones. Por ejemplo, un *software* puede crear una API para compartir la información de sus productos a otro servicio de *software* para su insumo y trabajo.  Su sigla significa interfaz de programación de aplicaciones. En este caso la palabra aplicación hace referencia a cualquier sistema y la interfaz sería el protocolo de comunicación entre dos servicios de dos sistemas.  La arquitectura de una API normalmente es cliente – servidor. La solicitud es enviada por el cliente hacía la API que estaría en el servidor y puede funcionar de cuatro formas diferentes:   * API SOAP * API RPC * API WebSocket * API Rest   Actualmente en la industria las API están cargadas en la nube, por ejemplo, en la nube de Amazon, Azure o cualquier otro servicio en donde el usuario hace las peticiones. Tenerlas en la nube hace que la empresa solo se preocupe por el desarrollo de la API y la comunicación, la seguridad y el control de tráfico. |

|  |  |
| --- | --- |
| Tipo de recurso | Cajón de texto de color |
| Los diagramas UML se caracterizan porque permiten una comunicación fluida con el cliente, comunican visualmente las ideas y, sobre todo, son precisos y eliminan la ambigüedad.  Existen diagramas que modelan la estructura y otros que modelan el comportamiento del *software*. A través de este modelamiento se comunican las ideas de forma certera; el cliente aprueba y se puede avanzar de forma correcta en el proyecto. | |

**Síntesis**

|  |  |
| --- | --- |
| **Tipo de recurso** | Síntesis |
| Desarrollo de aplicaciones web *full stack*  Síntesis: análisis, diseño y arquitectura de *software* | |
| **Introducción** | En la siguiente síntesis tendrá una mirada resumida de su ruta de conocimiento a través de este componente. |
| Análisis, diseño y arquitectura de *software*  Análisis  Metodologías de desarollo de *software*  Tradicionales  Ágiles  Diseño  Diseño orientado a objetos  Diagramas UML  Arquitectura  Diagrama de base de datos  API | |

**Actividad didáctica**

|  |  |  |
| --- | --- | --- |
| **Tipo de recurso** | Actividad didáctica. Opción múltiple | |
| Evaluar las temáticas estudiadas en el componente formativo. | | Especificar la imagen que acompañará el texto  Arquitecto de software diseñando un modelo UML en una pantalla. Ilustración 3D  **Imagen 1:** 228125\_i37 |
| **¿Cuál de los siguientes NO es un rol dentro de la Metodología SCRUM?** | |  |
| *Team* | | *Product Owner* |
| *ScrumController* (correcta) | | *ScrumMaster* |
| **¿Cuál de las siguientes NO es una fase de la metodología XP?** | |  |
| Planificación | | Simulación (correcta) |
| Pruebas | | Diseño |
| **¿Cuántas fases tiene la Metodología XP?** | |  |
| 5 (correcta) | | 4 |
| 3 | | 6 |
| **La metodología RUP es más adaptable a proyectos de:** | |  |
| Mediano plazo. (correcta) | | Corto plazo. |
| Largo plazo. | | Ninguna de las anteriores. |
| **¿Cuál de las siguientes NO es una fase de la metodología RUP?** | |  |
| Construcción | | Elaboración |
| Iniciación | | Potenciación (correcta) |
| **¿Cuál de los siguientes documentos NO es un artefacto de SCRUM?** | |  |
| *Burndown Chart* | | *Scrum Cost* (correcta) |
| *Sprint Backlog* | | *Product Backlog* |
| **El desarrollo rápido de aplicaciones (RAD) es una variación de:** | |  |
| XP | | Cascada (correcta) |
| JAD | | Métodos y prácticas |
| **El RAD es un lenguaje de desarrollo:** | |  |
| Ligero (correcta) | | Viable |
| Pesado | | Amigable |
| **Es parte de un diagrama de bases de datos.** | |  |
| Entidades (correcta) | | Objetos |
| Clases | | Herencia |
| **No es una relación entre clases en el diagrama de clases.** | |  |
| Herencia | | Asociación |
| Intervención (correcta) | | Composición |

Retroalimentación positiva:

¡Muy bien! Tiene claros los conceptos estudiados en el componente formativo.

Retroalimentación negativa:

Lo sentimos, algunos conceptos estudiados no se encuentran claros. Lo invitamos a repasar de nuevo el componente formativo y realizar de nuevo la actividad.

**Material complementario**

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de recurso | Material complementario | | |
| Tema | Referencia APA del material | Tipo | Enlace |
| Elementos de la estructura arquitectónica | Lucid Software Español. (2019). *Tutorial – Diagrama de Clases UML* (video). YouTube. <https://www.youtube.com/watch?v=Z0yLerU0g-Q> | Video | <https://www.youtube.com/watch?v=Z0yLerU0g-Q> |

**Glosario**

|  |  |
| --- | --- |
| **Tipo de recurso** | Glosario |
| API: | interfaz de programación de aplicaciones. |
| Atributo: | característica relevante de un objeto. |
| Clase: | entidad principal del mundo real que agrupa varios objetos con atributos en común y métodos en común. |
| Método: | es una función de código que realiza un comportamiento especifico de un objeto. |
| Objeto: | entidad del mundo real que representa la instancia de una clase. |
| Parámetro: | especifica el tipo de dato que recibirá un método. |
| UML: | lenguaje unificado de modelado. |
| Visibilidad: | valor que tendrá un atributo para conocer quién puede alcanzarlo en el sistema. |

**Referencias bibliográficas**

|  |  |
| --- | --- |
| **Tipo de recurso** | Bibliografía |
| Agilemanifesto.org. (s.f.). *Manifiesto por el Desarrollo Ágil de Software.* [*https://agilemanifesto.org/iso/es/manifesto.html*](https://agilemanifesto.org/iso/es/manifesto.html) | |
| Carrizo, D., & Alfaro, A. (2018). Método de aseguramiento de la calidad en una metodología de desarrollo de software: un enfoque práctico. *Ingeniare. Revista chilena de ingeniería*, *26*(1), 114-129. | |
| Guedes, G. T. (2018). *UML 2-Uma abordagem prática*. Novatec Editora. | |
| Jabangwe, R., Edison, H., & Duc, A. N. (2018). Software engineering process models for mobile app development: A systematic literature review. *Journal of Systems and Software*, *145*, 98-111. | |
| Montero, B. M., Cevallos, H. V., & Cuesta, J. D. (2018). Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software. *Espirales revista multidisciplinaria de investigación*, *2*(17), 114-121. | |
| Silva, A., Ledezma, E., Castorena, J., Domínguez, A., & Riojas, A. (2018). Utilidad del Lenguaje Unificado de Modelado (UML) en el desarrollo de software profesional dentro del sector empresarial y educativo. *Ciencia Cierta revista de divulgación científica*, *56*. | |
| Velásquez, S. M., Montoya, J. D. V., Adasme, M. E. G., Zapata, E. J. R., Pino, A. A., & Marín, S. L. (2019). Una revisión comparativa de la literatura acerca de metodologías tradicionales y modernas de desarrollo de software. *Revista Cintex*, *24*(2), 13-23. | |
| Zumba, J. (2018). Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software. *INNOVA Research Journal*, Vol 3, No. 10, 20-33. | |