**Clase 1:**

**METODO MIDAS**

Es un método de desarrollo de hipermedia basado en MDA. Su proceso de desarrollo es el siguiente:

1. Modelado de requisitos. Se capturan los requisitos a través de casos de uso.
2. Modelado Conceptual. Se construye un modelo conceptual OO, independiente de la plataforma.
3. Modelado Navegacional. El modelo conceptual se representa mediante un esquema XML y se transforma a un modelo navegacional.
4. Generación del modelo de datos. El modelo conceptual es transformado al modelo objeto relacional
5. Generación del modelo de presentación. El modelo navegacional es transformado a un documento XML, que es la plantilla del modelo de presentación.

El modelado de requisitos funcionales en MIDAS es a través de casos de uso; estos son etiquetados como servicio funcional básico.

Después, se desarrolla un diagrama de actividad para conocer el orden de ejecución de los casos de uso.

De cada caso de uso funcional se obtiene un slice funcional en el diagrama navegacional, y de cada caso de uso estructural se obtiene un slice estructural. La principal aportación del método es su técnica para derivar, desde los caso de uso, estructuras navegacionales de tipo funcional o estructural.

El método, prácticamente, construye un modelo de proceso, al establecer un orden de ejecución de los casos de uso y una separación de éstos por usuario. En cuanto, al empleo de MDA, no se puede justificar MIDAS utilize ese espacio tecnológico, pues no se menciona que los metamodelos estén definidos con MOF, ni que las transformaciones se hagan con un lenguaje de transformación de modelos.

**Métodos MDE y MDA para desarrollo de hipermedia**

Las características principales de una implementación de MDE son definir los modelos y metamodelos a través de un lenguaje de meta modelado, facilitar la construcción de modelos a través de servicios de las meta clases, y transformar los modelos en otros con lenguajes de transformación especializados.

La mayoría de los métodos de desarrollo de hipermedia son dirigidos por modelos.

Métodos como MIDAS, OO-H, OOHDM, OOWS y WebML generan código a través de sus herramientas CASE. Sin embargo, la representación de sus modelos y las transformaciones son parte del código embebido en sus herramientas. Es decir, son dirigidos por modelos, pero no se pueden catalogar como MDE.

**XML-XSLT**

Cuando se usa XML como espacio tecnológico de MDA, los modelos y metamodelos se definen como documentos XML y el lenguaje de transformación empleado es XSLT .XSLT, o Transformaciones del lenguaje extensible de hojas de estilo es un lenguaje para transformar documentos XML a otros documentos XML o a formatos tales como HTML, PDF, SVG, o texto, etc.

XSLT es además un estándar del W3C. Una transformación XSLT consiste en un conjunto de reglas, cada una de las cuales tiene un patrón de entrada, que sirve para identificar los nodos del documento de entrada a los que tiene que aplicarse la regla.

Durante una transformación XSLT, el árbol del documento XML de entrada es recorrido. Al visitar un nodo, se evalúa si satisface el patrón de entrada de alguna regla; si lo cumple, entonces se debe generar la salida de acuerdo a la regla. El orden de aplicación de las reglas no es importante, y suele haber un conflicto de resolución si varias reglas tienen el mismo patrón de entrada. La transformación continúa con el recorrido del árbol, y aplicando las reglas subsecuentes.

XSLT utiliza el lenguaje XPath para identificar los nodos del documento de entrada. XPath puede seleccionar nodos navegando el árbol en cualquier dirección, y aplicando predicados basados en la posición, contenido y atributos de los nodos. Las expresiones XPath son usadas para seleccionar el conjunto de nodos de entrada, para evaluar condiciones y para calcular valores que serán usados en los nodos del árbol de salida.

XSLT se puede aplicar a la transformación de modelos mediante la serialización de modelos UML o MOF en XMI (Intercambio de Metadatos en XML). Sin embargo, la mayor desventaja de este enfoque radica en que la transformación se define a un nivel sintáctico. Es decir, un modelo es una instancia del esquema XML que es equivalente a un metamodelo MOF.

La transformación entre los modelos no se establece mediante la transformación de los elementos del metamodelo, sino como la de elementos de un documento XML a otro. O sea, un nodo del documento que se encuentra en cierta posición, se transforma en un conjunto de nodos XML del documento destino, y se colocan en cierta posición del árbol del documento destino.

Si se quiere transformar un concepto inexistente en el metamodelo origen o en el destino XSLT no lo informa. Además, en XSLT las transformaciones dependen de la profundidad de los modelos, lo cual, a su vez depende de la definición de los metamodelos.

El formato XMI es muy difícil de leer y entender. Las referencias en un modelo XMI indican la posición de los elementos del modelo de forma diferente al estándar XML, esto hace muy complicado escribir las transformaciones.

Como en XSLT todo es relativo a la posición entonces no se tienen en cuenta las relaciones semánticas, que se indican en el metamodelo, de los elementos Arquitectura Dirigida por Modelos MDA es un marco de trabajo cuyo objetivo central es resolver el problema del cambio de tecnología e integración en el desarrollo de sistemas software.

La idea principal de MDA es usar modelos, de modo que las propiedades y características de los sistemas queden plasmadas de forma abstracta, y por tanto, los modelos no serían afectados por tales cambios.

MDA define dos clases de modelos: independientes de la plataforma (Platform Independent Model, PIM) y específicos de la plataforma (Platform Specific Model, PSM). Las definiciones de la guía de MDA son las siguientes: Plataforma. Una plataforma es un conjunto de subsistemas y tecnologías que provee un conjunto coherente de funcionalidades a través de interfaces y unos patrones específicos de uso, los cuales pueden ser empleados por cualquier aplicación sin que ́esta tenga conocimiento de los detalles de cómo la funcionalidad es implementada.

PIM. Un modelo independiente de la plataforma es una vista de un sistema desde un punto de vista independiente de la tecnología.

PSM. Un modelo específico de la plataforma es una vista de un sistema que depende de la tecnología donde se ejecutaría el sistema.

Los PIM son más abstractos que los PSM y éstos más que el código.

Por tanto, una transformación horizontal es de PIM a PIM o de PSM a PSM, y una transformación es vertical es de PIM a PSM o de PSM a código.

**Niveles de abstracción en MDA**

El desarrollo de un sistema de acuerdo al marco de trabajo de MDA se inicia con la construcción de un PIM.

Después el PIM es transformado en uno o más PSM. Por último, el código es generado a partir de los PSM. La operación fundamental en MDA es la transformación de PIM a PSM.

**Arquitectura dirigida por modelo**

Requiere como entrada un PIM, y tiene como salida un PSM. La función, además, recibe información adicional del usuario, a través del modelo de marcado, que sirve para indicar qué reglas aplicar, cómo transformar cierto elemento del modelo origen, e indicar decisiones de diseño.

Los PIM y PSM se marcan antes de la transformación, idealmente éstos no deberían contaminarse con las marcas, por eso el patrón recibe el modelo de marcado como entrada.

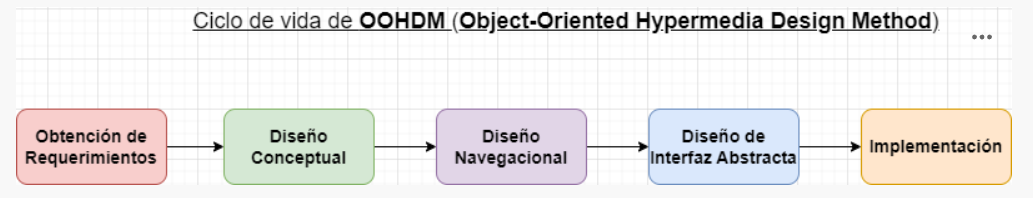
Sin embargo, lo más común es utilizar tagged values, por lo que el patrón de transformación real es de PIM-marcado a PSM. Por ejemplo, en un diagrama de clases se pueden marcar las clases persistentes con la etiqueta persistente; lo cual indicaría que esas clases se quieren transformar en tablas de un PSM que representa una base de datos. La guía de MDA dice que los modelos PIM y PSM se expresan como modelos UML y que las transformaciones deben automatizarse al máximo posible. La guía menciona como alternativas de transformación la directa (vía programación), el uso de patrones y el meta modelado. Desde el punto de vista de MDE también interesan las transformaciones de PIM a PIM y de PSM a PSM.

Las versiones iniciales de MDA sirvieron como base de MDE, que generalizó MDA, y que define las transformaciones en el contexto del meta modelado. En MDE la forma más aceptada de definir los modelos es a través del meta modelado, y las transformaciones a través de lenguajes de transformación de modelos. En cambio, en la propuesta original de MDA el meta modelado no es una condición necesaria.

Las versiones posteriores de MDA incorporaron después las ideas de MDE, que dieron lugar a MetaObject Facility (MOF) y Query Views Transformations (QVT).MDA es una realización de MDE basada en las tecnologías y estándares de OMG.

**Práctica:**

* **Analizar y desarrollar el CICLO DE VIDA de la Metodología de Desarrollo Web OOHDM**

****

**Clase 2:**

**Usabilidad y Accesibilidad**

La usabilidad y la accesibilidad son dos lentes ligeramente diferentes para evaluar la experiencia del usuario. Se puede ser fuerte en un área y débil en otra. El uso de cualquiera de los métodos por sí solo puede resultar en una representación inexacta de la experiencia del usuario de su sitio. Evaluar su sitio web teniendo en cuenta la facilidad de uso y la accesibilidad para todos los usuarios brinda la mejor experiencia de usuario posible.

**¿Qué es la usabilidad?**

La usabilidad se relaciona con la facilidad de uso de las cosas. En general, la usabilidad se mide en función de cinco criterios: recuerdo, eficiencia, errores, capacidad de aprendizaje y satisfacción (MEELS). Para respaldar estos criterios, pregúntese lo siguiente al evaluar la usabilidad de un sitio web:

* ¿Qué tareas esperan completar los usuarios usando el sitio?
* ¿Qué tan fácil es para alguien completar estas tareas?
* ¿Qué escenarios de prueba evaluarían la realización de estas tareas?
* ¿Qué datos debe registrar para capturar en la evaluación de esas tareas?
* ¿Qué tan satisfecho está el usuario con los pasos necesarios para completar las tareas?

Cuando haya respondido estas preguntas, uno debería preguntarse qué y cómo deberían cambiar las cosas para mejorar la experiencia del usuario. Por lo general, las pruebas de usabilidad tradicionales no tienen en cuenta al usuario discapacitado. Sin embargo, creemos que tener en cuenta a todos los usuarios es vital para el éxito de su sitio.

**¿Qué es la accesibilidad?**

La accesibilidad se relaciona con la forma en que una persona con discapacidad usa algo. La Sección 508 requiere que todos los sitios web del gobierno sean accesibles para usuarios con discapacidades. La Sección 504 extiende estos requisitos de acceso a cualquier grupo que reciba fondos federales.

Los sitios accesibles brindan información a través de múltiples canales sensoriales, como el sonido y la vista. Este enfoque multisensorial permite a los usuarios con discapacidad acceder a la misma información que los usuarios sin discapacidad. Por ejemplo, si tiene un video en su sitio, debe proporcionar acceso visual a la información de audio a través de subtítulos sincronizados.

Sin embargo, recordar que proporcionar un canal secundario para cumplir con los requisitos de la Sección 508 no garantiza que los usuarios con discapacidades tengan una experiencia igual y positiva en su sitio. Debe diseñar su canal secundario teniendo en cuenta tanto la audiencia como el contexto.

Por ejemplo, si la imagen está en relieve, debe marcarla con un texto alternativo vacío, que le indica al lector de pantalla que la omita. Sin embargo, cuando una imagen transmite información, como un gráfico, se debe considerar lo siguiente:

* ¿Qué información transmite el texto alternativo?
* ¿Qué dice el texto que lo rodea acerca de la gráfica?
* ¿Cuál es el mensaje para llevar a casa en el gráfico?

La mala atención a la audiencia y el contexto reduce la experiencia del usuario para las personas con discapacidad. En consecuencia, probar estos canales secundarios se vuelve tan importante como probar los canales primarios.

**Mejores prácticas de usabilidad y accesibilidad**

Aunque muchos libros y artículos de usabilidad brindan el número ideal de usuarios para una prueba, rara vez abordan la importancia de la diversidad en los temas de prueba. Al seleccionar grupos de prueba, los probadores a menudo se enfocan en el "usuario normal". Sin embargo, esto se hace a expensas de grupos de usuarios más pequeños, como las personas discapacitadas.

Dejar a las personas discapacitadas fuera de las pruebas de usabilidad crea una brecha en la metodología de prueba. Por ejemplo, un nuevo menú de navegación de un sitio web podría funcionar bien con usuarios sin discapacidades y recibir buenas puntuaciones en todas las categorías de MEELS. Sin embargo, si el contraste de color es insuficiente, o el menú no está marcado para trabajar con lectores de pantalla, o la navegación basada en el teclado se realiza correctamente, los usuarios ciegos y con problemas de visión no podrán utilizarlo.

Al no realizar pruebas con personas con discapacidad, es posible tener un sitio web con un alto grado de satisfacción y una fuerte usabilidad para la población sin discapacidad. Si bien esta población ya puede ser el "usuario normal del sitio" deseado, su sitio web puede ser completamente inutilizable e inaccesible para las personas con discapacidades.

**Práctica:**

* **Defina usabilidad. Por qué es importante tenerla en cuenta?**

Se refiere al nivel de facilidad con que las personas pueden utilizar una herramienta particula, en nuestro caso el software de desarrollo web. Será impotante tenerla en cuenta para mejorar la satisfaccion y fidelidad de los clientes, como tambie la reduccion en costes de soporte.

* **Qué es la accesibilidad? Cite 3 diferencias con la usabilidad.**

La accesibilidad se refiere a la capacidad de un sistema, producto o entorno (en nuestro caso las págians web), para ser utilizado por personas con diversas capacidades, incluidas aquellas con discapacidades físicas, sensoriales, cognitivas o tecnológicas.

Diferencia 1: La acccesibilidad se enfoca en en el acceso y no en la facilidad de uso.

Diferencia 2: El alcance, ya que en la accesibilidad existen diversos aspectos técnicos y de diseño que permiten a personas con discapacidades interactuar con el sistema, como el uso de lectores de pantalla, compatibilidad con teclados, subtítulos en videos, y colores contrastantes.

Diferencia 3: El objetivo principal, en la accesibilidad se busca como objetivo es eliminar barreras, mientras que en la usabilidad es optimizar la experiencia del usuario.

**Clase 3:**

**PROGRAMACIÓN EXTREMA (XP)**

La programación extrema tiene una gran influencia en el funcionamiento actual de las industrias de software. TDD, programación en pares, revisiones de código se han convertido en los estándares de facto como parte de XP.

Incluso después de 30 años de su creación, la programación extrema sigue siendo relevante y Martin Fowler llama a la programación extrema la base de las prácticas ágiles.

Exploremos cómo y por qué...

*En los primeros días de la programación, los proyectos de software se ejecutaban solo con las mejores habilidades de las personas que los escribían. Un pequeño error y el código causa estragos.*

*Incluso sin errores, una vez que la base de código se vuelve lo suficientemente grande, es difícil mantener el código existente, y mucho menos agregar nuevas características.*

*Los programadores se esfuerzan al máximo para asegurarse que el código siga funcionando, cumpliendo los plazos, etc., mientras que los clientes/partes interesadas probablemente se preguntaron si los programadores están siendo perezosos y no cumplen.*

Muchos de estos problemas se volvieron más fáciles de resolver y los esfuerzos para hacerlo se redujeron significativamente. Llegó con algo llamado **programación extrema**.

Veamos cómo funciona y por qué Kent Beck pensó en él y cómo incluso después de 30 años de su creación sigue siendo relevante.

¿Qué es la programación extrema y cómo resolvió todo?

*La programación extrema es una práctica basada en Agile que básicamente garantiza un software de la mejor calidad en poco tiempo, teniendo en cuenta el bienestar de los programadores y el compromiso con los clientes/accionistas.*

**Historia**

Todo comenzó en C3 (proyecto de Compensación Integral de Chrysler), un proyecto de nómina en Chrysler que se considera el lugar de nacimiento de la Programación Extrema.

C3 tenía como objetivo reemplazar el sistema basado en COBOL heredado y el desarrollo inicial comenzó en SmallTalk, que se suponía que terminaría en 2 años. Un año y medio después... hay algunos problemas de rendimiento y se supone que Kene Beck los ayudará a resolverlos.

Kent Beck: *"¿Dónde están las pruebas que demostrarán que cuando hago las cosas más rápido, nada se rompe?"*

C3: *"En realidad, el código aún no funciona"*

Kent Beck sugiere 3 opciones:

* Sigue todo como está.
* Subcontratar el proyecto (lo cual es una pena con gente tan excepcional, Martin Fowler también trabajó como consultor en este proyecto)
* Dé a todos una semana libre, deseche todo lo que se ha hecho hasta ahora y comience de nuevo y creo que tendrá éxito.

Kent Beck se pone a cargo. Él estructura el proyecto en incrementos de 3 semanas donde el trabajo se divide en historias y al final de cada incremento una demostración para los usuarios y un plan para las próximas 3 semanas.

Luego, alguien sugiere que cada historia debería tener alguna prueba de aceptación. Entonces alguien sugiere programación en pareja, etc. Y así sucesivamente, al final del día, había formulado una programación extrema. Más adelante en su libro, Kent Beck estandariza esta forma de hacer las cosas y la llama **programación extrema.**

**Comprender por qué funciona la programación extrema**

La programación extrema no es más que un conjunto de valores, principios y prácticas.

Eso es todo.

**¿Por qué?** Mejorar la calidad del código y del producto teniendo en cuenta el bienestar de los desarrolladores.

Para elaborar una programación más extrema se adhiere a ciertos valores no negociables (que sabemos que producirán buena calidad). Para adherirnos a estos valores utilizamos ciertas prácticas y mientras usamos estas prácticas, tenemos ciertos principios/directrices que debemos seguir para lograr los mejores resultados posibles.

Para explicarlo mejor, para un atleta el valor es tener un gran cuerpo, practica: haz ejercicio diariamente y el principio es no usar esteroides.

¿Cómo garantizarían estos valores una mejor calidad del código y el bienestar del desarrollador?, podría preguntarse.

Exploremos uno por uno.

* **Comunicación:** como todo lo demás, para poder lograr los mejores resultados, todos deben estar en la misma página. Los equipos deben hablar, explicarse y entenderse entre ellos.
* **Comentarios:** una vez que tengamos la misma dirección de vez en cuando, debemos reevaluarnos a nosotros mismos a partir de los comentarios brindados por los clientes o miembros de su equipo. Esto asegura que nos dirigimos hacia el mejor resultado posible con la satisfacción personal y del cliente.
* **Simplicidad:** Las soluciones más elegantes no son las que son complejas sino las que son sencillas de entender.
* *“Es mejor hacer algo simple hoy y pagar un poco más mañana para cambiarlo” que “hacer algo más complicado hoy que quizás nunca se use de todos modos”.*
* Esto significa menos explicaciones al incorporar nuevos miembros y una fácil mantenibilidad de nuestro trabajo existente.
* **Valentía:** Expresar sus opiniones, discutir y tomar decisiones por el bien del equipo/producto/cliente. A menudo conduce a las mejores ideas que originalmente no formaban parte del plan o ni siquiera se consideraron.
* **Respeto:** estar abierto a los demás y considerar la opinión y los pensamientos de todos no solo anima a las personas a compartir grandes ideas, sino que también se asegura de que los demás lo escuchen.

Ahora que tenemos nuestros valores establecidos, exploremos los principios.

**Principios**

**Principios:** retroalimentación rápida, simplicidad, cambio incremental, aceptación del cambio y calidad

**Retroalimentación rápida:** obtenga retroalimentación con frecuencia para asegurarse de que está satisfaciendo las necesidades del cliente y reflejando cuando ha aprendido a través de la retroalimentación en el producto y su trabajo.

**Simplicidad:** Resuelve cada problema de la forma más sencilla. Haga solo lo necesario para implementar realmente la función. Cualquier cosa extra puede agregar complejidad y puede ser completamente obsoleta y difícil de cambiar más tarde.

**Sigue a YAGNI (No lo vas a necesitar) y DRY (No te repitas)**

**Cambio incremental:** tratar de hacer muchas cosas a la vez puede terminar introduciendo muchos errores, código no deseado y terminar sin funcionar a tiempo. Los grandes cambios a la vez están destinados a fracasar. Por lo tanto, cada cambio debe desglosarse en una serie de pequeños pasos. Esto ayuda a rastrear el progreso y las unidades pequeñas también se pueden probar de manera eficiente.

Esta filosofía se puede aplicar al proyecto, al diseño, a la arquitectura o también a los cambios en la plantilla.

**Aceptar el cambio:** se deben aceptar los cambios en el diseño, el proyecto y el plan. Esto podría dar una ventaja competitiva al cliente para realizar cualquier cambio con confianza en función del mercado/entorno y la tranquilidad general para que el equipo sea lo suficientemente competente como para aceptarlo.

**Trabajo de calidad:** Un equipo que trabaja bien, hace un producto valioso y se siente orgulloso de ello. El cliente en general tiene una gran puntuación de satisfacción al final.

Ahora veamos algunas prácticas que nos ayudan a adherirnos a nuestros valores teniendo en cuenta nuestros principios.

**TDD:** el desarrollo basado en pruebas es una práctica en la que primero escribe las pruebas y luego escribe el código. Esto asegura una retroalimentación inmediata. También mantiene el código nítido y simple, ya que cada vez que escribe solo escribe tanto código como sea necesario para pasar la prueba. Cuando haya terminado, también tendrá una red de seguridad para continuar refactorizando sin romper nada.

**Refactorización de código:** la refactorización nos permite mejorar continuamente el código. Ayuda a eliminar la redundancia, el código innecesario y aumenta la legibilidad. Mantener el código nítido y simple nos permite realizar cambios fácilmente incluso después de mucho tiempo.

**CI/CD:** Permite que el equipo colabore en diferentes partes del código sin depender de nadie. El código es enviado varias veces al día por diferentes miembros. Las pruebas automatizadas y de extremo a extremo aseguran que todo funcione correctamente.

Pequeñas liberaciones-CI y pequeñas liberaciones van de la mano. CI nos permite lanzar versiones incrementales del producto para recibir comentarios del cliente. Esto nos permite detectar errores y probar el código en producción. El cliente puede sugerir un cambio anticipado después de cada lanzamiento.

**Programación en pareja:** es una técnica en la que dos desarrolladores se sientan juntos y escriben código. Mientras uno escribe, los otros revisan el código, sugieren mejoras y buscan errores. Luego se intercambian los roles. Esto significa un intercambio de conocimientos más rápido y una mejor comunicación. Los desarrolladores tienden a aprender prácticas unos de otros, lo que ayuda a establecer el código estándar a nivel de equipo/organización.

**Revisiones de código:** se basan en el modelo de relaciones públicas en el que los desarrolladores generan una relación pública para cada historia/función en la que trabajan. Esto permite adherirse al estándar de código de nivel de organización. Se puede recomendar a los desarrolladores que realicen cambios, si los hubiera. Esto también asegura que haya un segundo par de ojos que escanee el código asegurándose de que haya suficiente cobertura de código, que no haya errores y que, en general, la calidad del código sea buena.

Las herramientas automatizadas, como la nube SONAR, ayudan en las revisiones de código.

**Semana de 40 horas:** Esto asegura el bienestar del desarrollador. Están bien descansados ​​y en su mejor momento cuando trabajan. Un equipo es tan bueno como sus miembros. Esto garantiza que no haya agotamiento y que los desarrolladores puedan mantener un buen equilibrio entre el trabajo y la vida.

**Cliente en el sitio:** el cliente debe ser parte del proceso de desarrollo para responder cualquier pregunta, establecer prioridades y resolver disputas, si las hubiera.

**El juego de planificación:** esta es una reunión que ocurre al comienzo de un ciclo de iteración. El equipo de desarrollo y el cliente se reúnen para discutir y aprobar las características de un producto. Al final del juego de planificación, los desarrolladores planifican la próxima iteración y el lanzamiento, asignando tareas para cada uno de ellos.

**Estándares de codificación:** tener un conjunto común de prácticas de codificación, usar los mismos formatos y estilos para la escritura de código permite a todos los miembros del equipo leer, compartir y refactorizar el código con facilidad, así como acelerar el aprendizaje para otros programadores.

**Propiedad colectiva del código:** esta práctica delega la responsabilidad del código a todo el equipo, donde cada miembro es igualmente responsable. Esto garantiza que todo el equipo conozca todo el código y sepa exactamente dónde colocar una determinada característica y evitar la duplicación.

**Metáfora del sistema:** la metáfora del sistema representa un diseño simple de modo que las personas nuevas puedan comenzar a trabajar en él sin perder demasiado tiempo examinando las especificaciones. Y en general, el código es legible.

**Práctica:**

* ¿Cuándo sería útil usar XP?

Extreme Programming (XP) es útil en proyectos con requerimientos cambiantes, donde los requisitos no están completamente definidos desde el principio y pueden cambiar frecuentemente. También es ideal para proyectos con alta incertidumbre técnica, que involucran tecnologías nuevas o desconocidas. Funciona bien en equipos pequeños y colaborativos, donde las prácticas como el trabajo en pareja y la programación en pareja pueden ser más efectivas. Es beneficioso en proyectos con alta visibilidad y necesidad de calidad, ya que XP pone un fuerte énfasis en la calidad del código mediante pruebas continuas y refactorización. La enseñanza mas importante de XP es el énfasis en las pruebas y en concreto en la práctica de **TDD**.

* ¿Cuáles son las ventajas de usar XP?

Retroalimentación rápida, simplicidad, cambio incremental, aceptación del cambio y calidad

* ¿Cuáles son los roles más importantes de XP?

Cliente, programador, probador, rastreador, entrenador y consultor.

* ¿Cuál es el ciclo vital de XP?

Planificacion del proyecto -> Diseño -> Codificacion --> Pruebas

* Si se aplica TDD primero se harán las pruebas y después se codifica.

**Clase 4:**

**SCRUM**

Scrum es uno de los marcos más populares para implementar la mentalidad ágil. Funciona en un sprint de una a cuatro semanas en el que diseña, construye y prueba una parte de su producto. Y luego preséntelo a sus partes interesadas y vea si necesita ajustar su producto. Y luego repites el ciclo una y otra vez. De esta manera, está construyendo su producto de forma incremental después de cada sprint. Y al final, obtienes tu producto final.

En Scrum, un **sprint** es un período de tiempo establecido durante el cual se debe completar un trabajo específico y prepararlo para su revisión. Entonces, ahora que conocemos la idea básica detrás de Scrum, profundicemos en lo que sucede en Scrum.

**Roles de Scrum**

Hay tres roles que se definen en Scrum.

1. **Product Owner:** es el representante comercial del equipo que define qué se debe hacer y en qué orden. Es un trabajo de tiempo completo. Los propietarios del producto establecen la visión antes de que comience cualquier trabajo. La visión es el mapa que lo guiará a usted y a su equipo a su destino.
2. **Scrum Master:** facilita la mayoría de las reuniones del equipo y se asegura de que el equipo se mantenga fiel a los valores y principios de Scrum. Protege al equipo para completar el trabajo sin distracciones y resuelve los problemas del día a día.
3. **El equipo:** el equipo incluye desarrolladores, evaluadores y todos. Se auto organizan y realizan la mayor parte del desarrollo del software.

**User Stories**

El Product Owner hablará con los ejecutivos, el equipo, las partes interesadas, los clientes, los usuarios y definirá qué es exactamente lo que se debe construir, lo que se denomina acumulación de productos. La cartera de productos es una lista de historias de usuarios que se priorizan y define lo que se debe hacer.

Las historias de usuario son las interacciones que un cliente/usuario tiene con nuestro producto. Cada interacción del usuario con nuestro producto nos dice cómo los usuarios van a utilizar nuestro producto. Este es el nivel de detalle que necesitamos saber antes de trabajar en el producto. El propietario del producto escribe estas historias como resultado de entrevistar a las partes interesadas y al representante de los clientes.

En Scrum, las historias de usuario de alta calidad se escriben utilizando el acrónimo INVEST.

Las buenas historias de usuario son **independientes**. Se puede entregar por separado de las historias de otros usuarios y tener valor por sí mismo.

Las buenas historias de usuario son **negociables** hasta que la historia se comprometa a trabajar. Se puede reescribir, cambiar o cancelar en cualquier momento.

Las buenas historias de usuarios son **valiosas**. Ofrece valor al propietario del producto, a la parte interesada y al cliente.

Las buenas historias de usuario son **estimables**. debe poder estimar el tamaño de la historia y los puntos de la historia. Eso significa que la historia es lo suficientemente descriptiva para que sepas lo que hay que hacer para terminarla. Sólo entonces puedes entender el esfuerzo requerido.

Las buenas historias de usuario son **pequeñas**. La historia es lo suficientemente pequeña como para completarse en un sprint.

Las buenas historias de usuario son **testeables**. La historia proporciona suficiente información para que pueda desarrollar pruebas para ella.

Las historias de usuario son una herramienta táctica que utilizan los equipos de scrum para definir el trabajo de su producto. Escribir una excelente historia de usuario puede ser un desafío, pero con la práctica se vuelve más fácil con el tiempo.

**Criterios de Aceptación (AC)**

Las buenas historias de usuario también deben tener Criterios de aceptación, AC para abreviar. El AC es la herramienta más poderosa que tiene el equipo para reducir los esfuerzos para terminar la historia.

Es una lista de verificación que determina si todos los parámetros de una historia de usuario están completos, probados y funcionando antes de que el desarrollador pueda marcar la historia de usuario como "terminada". Se deben cumplir todos los criterios para garantizar que la historia de usuario funcione según lo planeado y probado.

Ejemplo: Para crear un perfil de usuario en un sitio de comercio electrónico, el AC puede ser

* El nombre del cliente se captura y se guarda.
* El correo electrónico del cliente se captura y guarda.
* El número de teléfono del cliente es capturado y guardado.
* La contraseña del cliente se captura y se guarda.
* La dirección de envío del cliente se captura y guarda.

Cada vez que se escribe una historia de usuario, el propietario del producto y el equipo colaboran para determinar el AC de la historia. Cada historia tiene un conjunto único de AC.

Los criterios de aceptación deben ser lo más explícitos posible para que todas las partes del equipo sepan qué se hace primero.

**Criterios de éxito**

A estas alturas, debería tener sus funciones y temas desglosados ​​en las historias habituales y también debe tener AC asignado a todas las historias de usuarios. Esta es su cartera de productos.

Su próximo paso es crear límites que su equipo pueda usar para completar el trabajo. Estas pautas incluyen la definición del equipo de "TERMINADO", la preparación de la acumulación y el establecimiento de la cadencia de sprint.

**Definition of Done**

En primer lugar, su equipo debe definir qué significa hecho para sus historias. A diferencia de AC, la definición de Listo es más general, debe cumplir con los requisitos mínimos para todas las historias en la cartera de pedidos.

Por ejemplo: su equipo puede decidir que una historia se puede llamar como TERMINADA solo cuando se revisa y prueba el código en el entorno de versión preliminar.

**Backlog Grooming**

Simplemente significa que el propietario del producto secuencia continuamente las historias en orden de valor. Cuanto más valioso sea el resultado de la historia, mayor será su acumulación. El propietario del producto debe asegurarse de que las cosas más valiosas se entreguen primero. De esa manera, incluso si se agota el tiempo, el valioso producto aún ha sido entregado.

Si hay historias más grandes que deben dividirse en más pequeñas para que puedan caber en un sprint, entonces el propietario del producto también trabaja en eso.

Todas estas actividades ocurren en una reunión o en una sesión, como la limpieza del trabajo pendiente.

**Sprint Cadence**

Finalmente debes establecer tu cadencia o duración de sprint. Scrum dice que Sprint puede tener una duración de 1 a 4 semanas con la preferencia preferida hacia una escala de tiempo más corta.

Recuerde que Scrums dice que debemos fallar y aprender rápido. Siendo ese el caso, debe establecer la longitud de Sprint más corta que pueda.

Cuando trabajo con equipos, generalmente recomiendo ir con alrededor de 2 semanas. Esta duración realmente ayuda al equipo a mantenerse enfocado en lo que sucede dentro del sprint.

Un sprint de dos semanas es la zona Goldilocks. En mi opinión.

Si su Sprint es demasiado corto, el equipo intenta entrar en pánico y la calidad se degrada. Si el sprint es demasiado largo, el equipo se relajará inconscientemente y su ritmo se degradará.

Establecer estos tres límites ayudará al equipo a comprender qué significa "Terminado" para las historias pendientes, qué valor tiene cada historia y en qué momento debe entregarse cada historia. Por lo tanto, ahora tienen un marco de ejecución perfecto para ejecutar.

**Reunión de planificación de Sprint**

Una vez que el equipo está listo para el sprint, se reúnen para una reunión llamada Reunión de planificación del sprint. En esta reunión, el equipo elige las principales historias en las que pueden trabajar en el Sprint existente. El propietario del producto revisa esas historias con el equipo y ayuda a responder cualquier pregunta o aclara cualquier cosa que no esté clara.

Más tarde, el equipo se reúne para asignar tareas a las historias, para descubrir qué deben hacer exactamente para construir el software, como diseñar la base de datos, desarrollar la interfaz de usuario y otros relacionados.

Una vez que el equipo haya asignado las historias, determinará cuántos sprints se necesitan para terminar las historias seleccionadas. Y entonces los equipos comienzan su ejecución.

El objetivo aquí es que todos en el equipo entiendan completamente la intención de la historia y los criterios de aceptación específicos para esa historia. También ayuda publicar la definición del equipo de hecho para todas las historias en la sala de reuniones.

**Reunión Standup**

El scrum master establece tu scrum diario. Esto también se conoce comúnmente como la reunión diaria de pie o standup para abreviar. Incluye a todos los miembros del equipo, incluido el propietario del producto y el Scrum Master.

Para que los scrums funcionen, se basa en las tres C. Colaboración Comunicación y Cadencia.

Ocurre a la misma hora todos los días. Por lo tanto, su maestro de scrum seleccionará el tiempo que funcione para todos.

Por lo general, se lleva a cabo frente al tablero de tareas del equipo, ya sea electrónico o físico. Este es el momento en el que cada miembro del equipo se mueve por el tablero dando una visión general de su progreso.

Durante la reunión, todos hablan sobre lo que hicieron ayer, lo que van a hacer hoy y, si hay obstáculos, piden ayuda.

Nadie debería estar entregando el golpe por golpe de sus actividades. Solo la descripción general.

Las reuniones diarias deben ser breves. Debe limitarse a 15 minutos. Puede ser más corto pero no puede ser más largo. Todo el equipo se pone de pie para mantenerlo rápido.

Scrum master dirige la reunión asegurándose de que cada miembro del equipo siga las reglas mencionadas anteriormente.

La conclusión es que su standup diario es uno de los no negociables del marco de scrum.

**Informes de Scrum**

Si está trabajando en un proyecto crítico, es bastante común que las partes interesadas pregunten a los miembros de su equipo cómo van las cosas. Esto es bueno, la gente está interesada en el resultado de tu proyecto.

Sin embargo, puede ser una distracción para su equipo, ya que algunos miembros del equipo tienen una perspectiva completa de cómo van las cosas y otros no.

Scrum aborda los desafíos mediante la publicación de radiadores de información. Un radiador de información es cualquier cosa que publique en el muro o en los sitios de equipo que ayude a todos a comprender lo que está haciendo y cómo va.

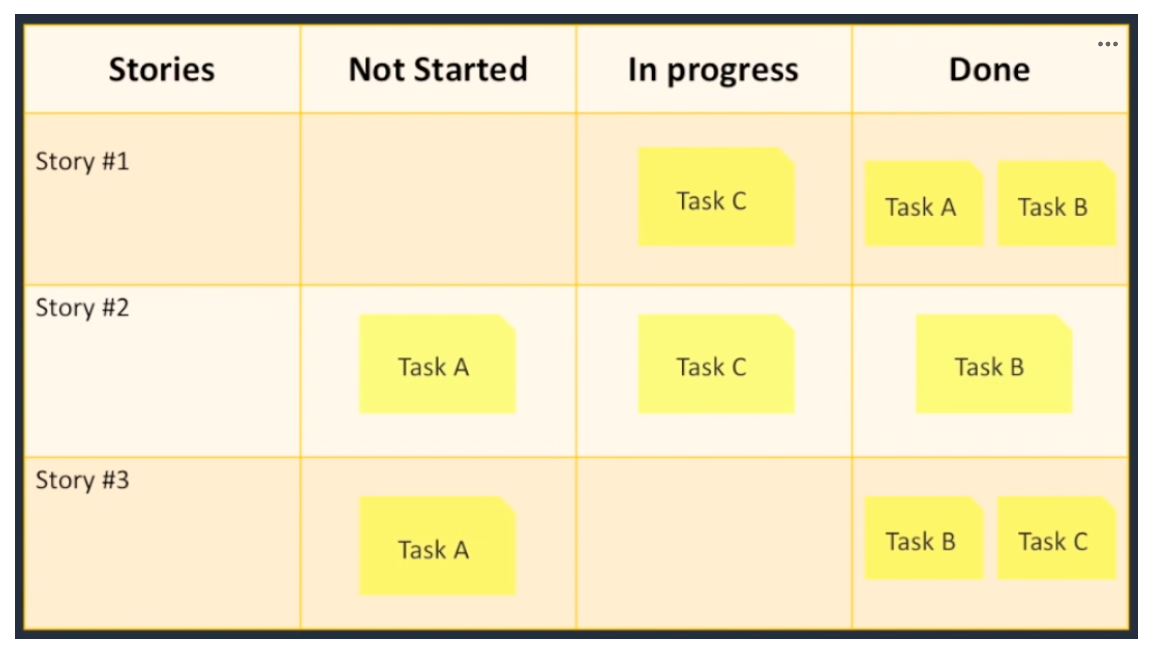
Como mínimo, los equipos publican sus tableros de tareas y su tabla quemada.

**Task Board**

El tablero de tareas puede adoptar varias formas en cualquier forma que desee, pero tiene algunos componentes clave que debe seguir.

Muestra las historias comprometidas en el sprint, las historias en su estado actual y las historias que se completaron.

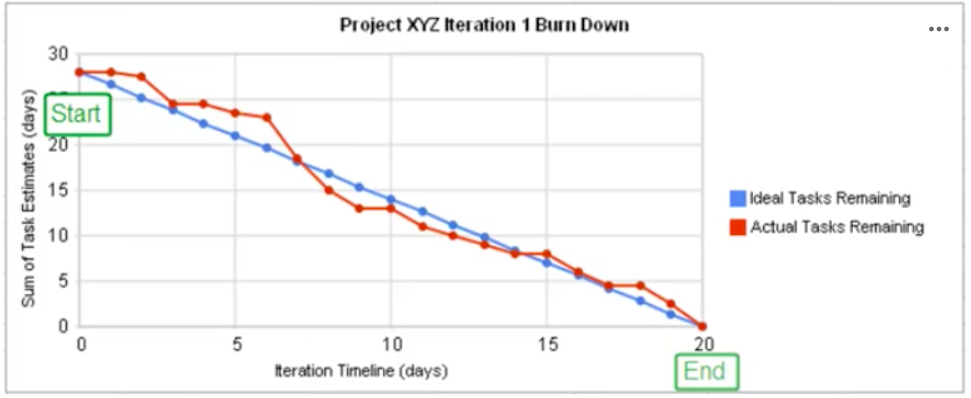
A continuación se muestra un ejemplo simple de cómo puede verse un tablero de tareas.



**Burndown chart**

Otra herramienta principal para compartir información sobre su progreso es el gráfico quemado de sprint. El equipo lo usa para medir qué tan bien se están ejecutando en el sprint.

Aquí hay un ejemplo de un sprint quemado.



Como puede ver, el quemado le dice exactamente dónde se encuentra en el sprint por día y cuánto trabajo se ha comprometido a hacer en todo el sprint por horas. Idealmente, se quemará de forma lineal.

Entonces, en este diagrama, el eje x es el tiempo que muestra los días del sprint, y el eje y es el total de algunas estimaciones de tareas.

La línea azul muestra cuál es la forma ideal de reducir la cantidad de trabajo. Comienza con el número total de trabajo y luego, al final del sprint, se reduce a cero.

La línea roja muestra la tarea real restante. Como puede ver, al principio, el equipo estaba bien y luego, en el medio, no estaban encaminados y finalmente, regresaron.

La línea real te dice exactamente cómo te está yendo en comparación con el ideal. Esta es una herramienta muy poderosa para mostrarle al equipo y a todas las partes interesadas cómo está progresando el equipo.

Fácil de crear, fácil de usar y entender. Estos métodos de informes de sprint serán de gran utilidad para su equipo y sus partes interesadas apreciarán tener un estado claro.

**Revisión de Sprint y retrospectiva**

Hay otras dos reuniones que ocurren al final del sprint.

**Revisión de sprint**, donde todo el equipo se reúne con las partes interesadas, con el cliente, y demuestra el trabajo que han realizado y obtienen la retroalimentación.

**Retrospectivas de Sprint**, donde hablan del proceso y no del producto. La mayoría de las discusiones ocurren sobre lo que salió bien en el último sprint, lo que no salió bien en el último sprint y cómo podemos hacerlo mejor. Esto satisface los principios detrás de Agile que dicen que el equipo necesita la oportunidad de reflexionar regularmente sobre cómo ser más efectivo y ajustar su comportamiento en consecuencia.

Finalmente, veamos cómo Scrum realmente respalda los principios ágiles:

1. Como puede ver, estamos construyendo de manera intuitiva, por lo que acepta el cambio.
2. Después de cada sprint, puede realizar cambios en su cartera de productos y cambiar sus productos en una dirección diferente.
3. También es compatible con una gran cantidad de reuniones de colaboración.
4. Apoya la mejora continua con la retrospectiva de sprint

Y la lista continúa, como podemos ver, al menos incorpora los principios clave de Agile.

**Práctica:**

* Defina SCRUM

Scrum es un marco ágil utilizado para el desarrollo de productos de manera incremental e iterativa. Funciona en ciclos llamados sprints, que duran de una a cuatro semanas. Durante cada sprint, el equipo de desarrollo diseña, construye y prueba una parte del producto, que luego es presentado a las partes interesadas para recibir retroalimentación y realizar ajustes si es necesario.

* ¿Qué ventajas existen a la hora de aplicar SCRUM?
* Scrum permite **cambios y ajustes rápidos** basados en la retroalimentación continua de las partes interesadas.
* A través de reuniones diarias y revisiones de sprint, todos los miembros del equipo y las partes interesadas tienen **visibilidad del progreso** y de los obstáculos.
* La retrospectiva de sprint proporciona una oportunidad para reflexionar sobre lo que ha funcionado y lo que no, fomentando la **mejora continua** del proceso y del equipo.
* Scrum permite la entrega rápida eincremental de partes del producto que **generan valor**, permitiendo que las partes interesadas vean resultados y reciban valor antes de que el producto esté completamente terminado.
* ¿Cuándo es mejor aplicar SCRUM?

Scrum es especialmente útil en proyectos donde los requisitos no están completamente definidos desde el principio y pueden cambiar a medida que el proyecto avanza. Es ideal para entornos dinámicos donde la adaptabilidad y la respuesta rápida al cambio son cruciales.

* Nombre los roles en SCRUM

**Product Owner:** es el representante comercial del equipo que define qué se debe hacer y en qué orden. Es un trabajo de tiempo completo. Los propietarios del producto establecen la visión antes de que comience cualquier trabajo. La visión es el mapa que lo guiará a usted y a su equipo a su destino.

**Scrum Master:** facilita la mayoría de las reuniones del equipo y se asegura de que el equipo se mantenga fiel a los valores y principios de Scrum. Protege al equipo para completar el trabajo sin distracciones y resuelve los problemas del día a día.

**Team Development:** el equipo incluye desarrolladores, evaluadores y todos. Se auto organizan y realizan la mayor parte del desarrollo del software.

**Clase 5:**

**¿Qué es Ágil?**

**La palabra "ágil" significa la capacidad de mantener las cosas en movimiento con facilidad y sin obstáculos. Su objetivo es mostrar la adaptabilidad y la respuesta al cambio , que es un concepto importante en este enfoque.**

En el proceso de desarrollo de software, por ejemplo, indica el buen funcionamiento del proceso, que en esta industria ha presentado un enfoque de trabajo diferente con algunos puntos fundamentales únicos.

**La metodología ágil se desarrolla y aplica principalmente en dos áreas: desarrollo de software y gestión de proyectos.**

La metodología ágil de desarrollo de software es un término general para un conjunto de marcos y prácticas, centrado en la idea de desarrollo iterativo, donde los requisitos y las soluciones evolucionan a través de la colaboración entre equipos multifuncionales autoorganizados. Este proceso permite que el equipo entregue mejores y más rápidos resultados. Scrum y Kanban son dos de las herramientas más utilizadas aquí.

La gestión ágil de proyectos se basa en el manifiesto Agile y parte de la premisa de que los proyectos se basan en personas que necesitan colaborar bien y tener conjuntos de habilidades que se complementan entre sí. Se trata menos de organigramas y más de reunir a profesionales y hacer que piensen en la mejor manera de hacer las cosas.

Cuando se utilizó por primera vez en 2001, en el contexto del manifiesto Agile, el término Agile se vinculó principalmente con el desarrollo de software. **Después de eso, el método comenzó a extenderse gradualmente a diferentes dominios e industrias** y la gestión ágil de proyectos se convirtió en una mentalidad generalmente preparada a aceptar y lidiar con los cambios en un ecosistema turbulento e incierto.

**El manifiesto ágil: 4 valores y 12 principios establecidos para adoptar cambios y flexibilidad**

En 2001, cuando crearon el Manifiesto Ágil, los 17 desarrolladores de software involucrados en el proceso se comprometieron con los siguientes cuatro valores:

**1. Individuos e interacciones sobre procesos y herramientas** . Este valor establece que las personas son quienes diseñan las herramientas y las utilizan en su beneficio para impulsar el desarrollo.

**2. Software de trabajo sobre documentación completa.**Agile acepta la importancia de la documentación, pero la considera una herramienta más que un propósito; por lo tanto, la mentalidad brinda a los desarrolladores de software lo que necesitan para trabajar, sin sobrecargarlos con información.

**3. Colaboración con el cliente sobre la negociación del contrato. La gestión ágil de proyectos ubica a los clientes en todos sus procesos** , desde la investigación de mercado hasta el despliegue y la entrega, para garantizar que el resultado final cumpla con sus demandas. A diferencia de este enfoque, la negociación solo se aplica antes del proceso. Una vez que las dos entidades involucradas, es decir, el cliente y el director del proyecto, han decidido los detalles del entregable, es difícil hacer cambios.

**4. Responder al cambio sobre seguir un plan.**Si bien el desarrollo de software tradicional no fomentó el concepto de cambio y se centró en planes detallados que debían respetarse por completo, **Agile respalda la idea de que los cambios son buenos ya que aportan valor agregado a los proyectos.**

En lo que se refiere a los principios Agile, **están destinados a apoyar una cultura que acoge cambios inesperados y gira en torno a los clientes** y sus comportamientos y necesidades, así como **los objetivos y el potencial del negocio.**

Estos son los 12 principios:

* Satisfacción del cliente mediante la entrega temprana y continua de software valioso;
* Dar la bienvenida a los requisitos cambiantes, incluso en etapas de desarrollo posteriores;
* Entregar software que funcione con frecuencia (semanas en lugar de meses);
* Cooperación estrecha y diaria entre socios comerciales y desarrolladores;
* Construir proyectos en torno a personas motivadas, en quienes se debe confiar;
* Conversación cara a cara, como la mejor forma de comunicación (coubicación);
* Software de trabajo como la medida principal de progreso;
* Desarrollo sostenible, capaz de mantener un ritmo constante;
* Atención continua a la excelencia técnica y al buen diseño;
* Simplicidad: el arte de maximizar la cantidad de trabajo no realizado;
* Las mejores arquitecturas, requisitos y diseños que surgen de equipos autoorganizados;
* El equipo está reflexionando sobre cómo ser más eficaz y se ajusta en consecuencia.

**¿Cómo funciona Ágil? Proceso, características y beneficios**

**Agile toma proyectos y los divide en incrementos más pequeños, en el marco de Scrum,** y entrega las tareas en ciclos conocidos como iteraciones, que tienen una duración recomendada de 2 semanas.

Antes de ponerse manos a la obra, tanto el cliente como el jefe de proyecto están **obligados a hacer una lista de “historias de usuario” o “objetivos”** que son obligatorios para el proyecto, evaluar cuánto tiempo toman y luego priorizar las mini tareas según su importancia. y urgencia.

**Esas asignaciones, una vez aprobadas, luego irán a Kanban hasta que se logren, verifiquen y eliminen,** lo que allana el camino para que ingresen nuevas tareas. Un proyecto ágil se considera terminado solo cuando no quedan incrementos para entregar.

1. **Ciclo de implementación de dos semanas** : La provisión de flexibilidad en el proceso de desarrollo es un ciclo de implementación de 2 semanas.
2. **Método de Verificación** : Proceso rápido de verificación paso a paso y correcciones inmediatas.
3. **Marco Scrum** : eliminación de actividades superfluas, es decir, la mejor implementación viva hoy en día es el marco Scrum.

El enfoque ágil convierte las fases tradicionales de desarrollo de software, como el análisis, el diseño, la codificación y las pruebas, en actividades continuas que coexisten a lo largo del desarrollo del proyecto.

**Este tipo de arquitectura en evolución permite a los agilistas mantenerse rentables y rentables, así como mantener la flexibilidad y dar forma a los proyectos a medida que avanzan.**

**Además, se basa constantemente en los comentarios de los clientes para que los agilistas logren evitar problemas.**

**4. Adaptativo al riesgo** : Agile admite la flexibilidad y está orientado a adaptarse al cambio, sin importar en qué etapa del proceso intervenga.

**5. Extremadamente pragmático.**La mejor medida para evaluar un proyecto para Agilistas es desarrollar un software que responda a las demandas del cliente en el menor tiempo posible.

**6. Documentación insignificante** : dividir los pasos más grandes en pequeños hitos con una documentación mínima.

Agile comienza con un proyecto básico construido y desarrollado con el tiempo con cada pequeño incremento en lugar de entregar la tarea completa de una sola vez.

**7. Enfoque colectivo** : el estilo de trabajo colaborativo está destinado a 10-12 programadores en colocación.

Se basa en la comunicación cara a cara, ya que los equipos suelen estar ubicados en la misma sala. **Esto reduce los tiempos de espera asociados con llamadas telefónicas, correos electrónicos, etc.**Para maximizar la eficiencia, cada equipo incluye un representante del cliente, es decir, el equivalente a un propietario de producto Scrum.

**8. Diversidad en los miembros del equipo.**

Un equipo ágil está compuesto por diferentes profesionales: gerentes de proyecto, diseñadores de UX, desarrolladores, analistas comerciales, analistas de control de calidad, etc. Pero una vez que los reúne, estos roles se desvanecen a medida que cada uno de ellos contribuye utilizando su propia experiencia para brindar lo mejor de la equipo.

Como dicen los investigadores, la metodología ágil pacta con un ciclo de vida sostenido y más corto con resultados espectaculares. Esto no solo proporcionaría un mejor enfoque para el trabajo, sino que de hecho ahorra tiempo y el sudor que ponen en el trabajo siguiendo el enfoque lineal.

**Agile vs. Waterfall: ¿Cómo elegir uno que se adapte mejor a su negocio?**

A diferencia de la gestión de proyectos Agile, que tiene un enfoque iterativo e incremental, el **modelo de cascada, también conocido como modelo de ciclo de vida secuencial lineal, se rige por secuencias, ya que para pasar a una nueva etapa, el equipo debe tener éxito. completado el anterior.**Este método también se conoce como **enfoque basado en un plan,** ya que se apega en gran medida a un solo plan.

**1. Presencia del cliente: Agile requiere que los clientes sean una parte activa de su mentalidad** , mientras que Waterfall solo depende de los clientes al completar cada hito.

**2. Ámbito:** en lo que respecta al alcance real, Agile se diferencia de Waterfall en que **se adapta a los cambios y es reconocido por adaptar el alcance y características tales como el tiempo, los presupuestos e incluso los objetivos. Funciona en un ámbito limitado, no más allá.**Waterfall, por otro lado, es sensible a los cambios y funciona muy bien cuando las limitaciones contractuales o criterios similares infringen el alcance.

**3. Priorización:** Agile atribuye diferentes niveles de prioridad a varios objetivos para garantizar que **los mejor calificados se finalicen primero. Se enfoca en completar primero la parte necesaria y así reduce los riesgos de falla** .

En comparación, Waterfall tiene un enfoque de todo o nada, ya que se basa únicamente en negociaciones. No acepta desafíos a menos que se describan en el contrato y, por lo tanto, disminuye la flexibilidad y aumenta las posibilidades de falla, especialmente en un entorno inestable, poco confiable y ambiguo.

**4. Estilo de trabajo:** Agile se basa en la **colaboración, las reuniones diarias y la comunicación cara a cara** ; mientras que Waterfalls hace uso de roles clásicos que se reúnen en etapas específicas a través de coordinadores de proyectos.

**5. Financiamiento:** esta es la parte más importante que preocupa a las empresas cuando intentan cambiar el enfoque. **La metodología Agile funciona en la financiación no fija** y se paga solo de acuerdo con los hitos terminados. Esto es beneficioso para los clientes, ya que tienen la opción de cancelar, suspender o realizar un pago cuando se completa un único hito. Por otro lado, Waterfall está restringido a proceder con financiación fija.

**6. En pocas palabras: ¿cuál es mejor?**Después de analizar las ventajas y desventajas tanto de Agile como de Waterfall, se puede concluir que **la respuesta radica en su propia ideología y proceso comercial.**

Si desea construir un puente o una infraestructura, por ejemplo, se recomienda adoptar un enfoque en cascada, ya que primero necesitaría trabajar en un plan con los clientes y luego ceñirse a él. Además, en tal escenario, es imposible reiterar y volver a una etapa anterior, que Agile requiere.

**Tanto los desarrolladores como la administración están adoptando Agile debido a un mejor rendimiento**

Agile se ha convertido en la norma en el desarrollo de software, ya que la investigación muestra que más del 85 % de los desarrolladores utilizan la metodología ágil, en comparación con un poco más del 15 % en Waterfall.

Además, según PWC , los proyectos que implementan Agile se consideran un 28% más exitosos que los tradicionales . Las razones para adoptar tal mentalidad incluyen : acelerar la entrega de software (74 % de los encuestados), mejorar la moral del equipo (34 %) y reducir los riesgos del proyecto (28 %).

La metodología ágil se apalanca tanto en el desarrollo de software, ya que ha sido su primer uso desde que se creó el manifiesto, como en la gestión. **Los principios de este enfoque se utilizaron tanto que se expandieron incluso en la industria del marketing, es decir, el marketing ágil,** que utiliza la recopilación y el análisis de datos para identificar oportunidades e implementar soluciones en tiempo real. Este tipo de flujo de trabajo permite a los especialistas en marketing generar ideas que pueden evaluarse y probarse rápidamente.

A nivel mundial, corporaciones como **Apple, Nokia, Ericsson e Intel están utilizando Agile en el desarrollo, la gestión de proyectos y el marketing para fomentar la innovación, la creatividad, la flexibilidad y la colaboración.**

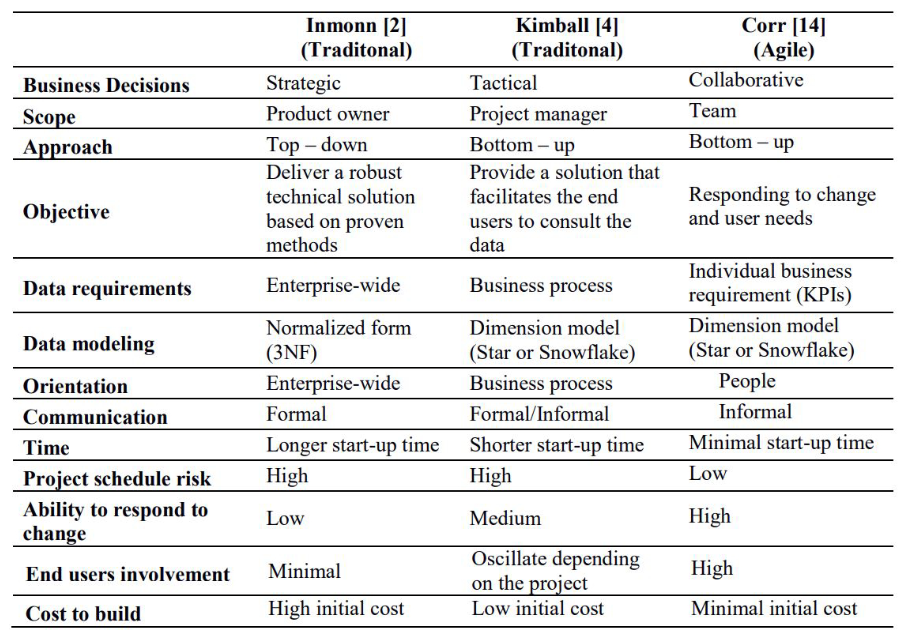
**2 estudios de casos del mundo real: desarrollo de software y marketing**

Un estudio de caso basado en la investigación de Mercado et al. 2018 , donde una empresa desarrolladora de software internacional en México adoptó un enfoque ágil en soluciones de Business Intelligence (BI), con respecto al diseño y desarrollo de almacenes de datos.

Después de comparar los 3 métodos diferentes, que se ven a continuación, la empresa decidió optar por la tercera opción y resultó ser una gran elección.

**El enfoque ágil se centró en las necesidades de los usuarios y pudo responder fácilmente a los cambios.**Además, se **realizaron reuniones con el equipo de trabajo, es decir, stakeholders, usuarios de BI, desarrolladores de ETL, analistas de negocio, entre otros, con el fin de recopilar los requerimientos de información durante todas las etapas del proyecto enfatizando la comunicación ininterrumpida y el trabajo colaborativo.**

Por estas razones, la implementación de ágil logró ayudar a mejorar la comprensión de todos los interesados ​​hacia el nuevo sistema de BI, resolver los problemas diarios de los usuarios e impactar positivamente en la mejora de los procesos de la cadena de valor de la empresa.



Otro caso de estudio. Netflix, el servicio de transmisión, está implementando y aprovechando la metodología ágil con éxito en términos de creación de contenido, gestión de relaciones con el cliente y marketing, lo que podría ser un buen ejemplo para usted:

**1. Netflix no lanza episodios piloto.**En lugar de invertir una gran cantidad de tiempo y recursos en un episodio que podría tener éxito o no, Netflix se basa en la recopilación y el análisis de datos para predecir los comportamientos de los usuarios a fin de aumentar las tasas de éxito y crear programas que atraigan el máximo tráfico.

**2. Netflix está integrando constantemente los comentarios de los consumidores.**Netflix entiende dónde se posicionan sus usuarios y aplica cambios en tiempo real para ganar ganar.

**3. Netflix se transmite con éxito en 900 dispositivos diferentes** , lo que significa que la empresa hace un uso perfecto de las pruebas entre navegadores según las demandas de los consumidores.

**4. Netflix prefiere hacer lanzamientos frecuentes y consistentes.**La plataforma crea contenido que se publica en ciclos más cortos, con el apoyo de la gestión de proyectos Agile.

**5. Netflix trabaja con objetivos Agile detallados que involucran a cada espectador** , mediante la creación de contenido, recomendaciones y análisis personalizados para cada usuario según su historial de navegación y el recorrido del cliente.

**Consideración futura y próximos desafíos para los admiradores ágiles**

La investigación indica que el 76% de las empresas crecerán para 2020, especialmente las empresas que utilizan una metodología ágil. El hecho es que, la mayoría de las veces, la razón por la que las empresas se enfrentan a dificultades radica en el enfoque básico en sí mismo (Williams, 2010). Esto puede ser una pista para todos aquellos que quieran sobrevivir en un mundo donde cada empresa se ve afectada por cambios rápidos, masivos y complicados: cada vez es más difícil encontrar otro método que resulte más ingenioso y versátil que Agile , dijo Steven Denning. para Forbes.

**A pesar de algunas de sus deficiencias,** por ejemplo, demandas de un alto nivel de compromiso de cada miembro del equipo, así como de los clientes y documentación menos detallada para facilitar los procesos necesarios como el onboarding, **la metodología Agile sigue siendo una revolución tecnológica y está mejorando hasta el día de hoy. .**

**La buena noticia es que gracias al avance de la Inteligencia Artificial (IA), la automatización de este enfoque tendrá más implicaciones que antes.**

A partir de esto, podemos predecir que el proceso de agilidad será aún más fluido, ya que la mayoría de las tareas tradicionales se realizarán con la ayuda de la automatización y, por lo tanto, el tamaño del equipo será más pequeño, lo que impulsará aún más la comunicación y la eficiencia (Williams, 2010).

**Práctica:**

* ¿Cuándo es propicio usar Metodologías Ágiles?

Las metodologías ágiles son más útiles en entornos donde se requiere flexibilidad, adaptabilidad y colaboración constante con el cliente. Son propicias en proyectos con alta incertidumbre, donde los requisitos pueden cambiar frecuentemente, o donde el producto final no está claramente definido desde el inicio. También son ideales en equipos multidisciplinarios que necesitan ciclos de desarrollo cortos y entregas frecuentes de valor.

* ¿Desarrolle brevemente cada metodología ágil definida?

**Scrum:** Es una metodología que organiza el trabajo en ciclos cortos llamados "sprints" (generalmente de 2 a 4 semanas). Cada sprint tiene objetivos claros y al final se entrega un incremento funcional del producto. Se caracteriza por roles definidos como el Product Owner, Scrum Master y el equipo de desarrollo.

**Kanban**: Es una metodología visual centrada en la gestión del flujo de trabajo. Utiliza un tablero donde las tareas se dividen en columnas como "Por hacer", "En proceso" y "Hecho". Es muy flexible y no tiene iteraciones fijas.

**XP (Extreme Programming):** Se enfoca en mejorar la calidad del software y la capacidad de respuesta ante cambios de los clientes. Sus prácticas incluyen el desarrollo incremental, programación en parejas y pruebas continuas.

**Lean Development:** Se basa en la eliminación de desperdicios y la entrega rápida de valor al cliente, enfocándose en la mejora continua y la optimización del flujo de trabajo.

**Clase 6:**

Comenzaremos esta clase con algunos conceptos que tratarán de dejar en claro la diferencia entre los sistemas monolíticos y los sistemas orientados a servicios.

La interconexión entre sistemas o utilizar servicios que ofrecían otros sistemas desde nuestra aplicación era bastante difícil.

* Las empresas creaban sus propios sistemas con sus datos residentes en el mismo sitio.
* Eran métodos “monolíticos”, con necesidades casi nulas de integración.
* Necesitaba **una integración muy profunda, generalmente de punto a punto**

Con el tiempo surgieron sistemas especializados integrando distintos servicios que consumían las áreas de una organización. (ERP, CRMS, CMS..)

Estos sistemas eran desarrollados por diferentes empresas y tecnologías donde se encontraban problemas de integración e interoperabiidad.

Para ayudar a solucionar este problema surgieron varias tecnologías:

•**DCOM** (Distributed Component Object Model) utilizando **RCP** (Remote Procedure Call) que soporta **mensajes muy básicos**.

•**CORBA** (Common Objet Request Broker Architecture) que **requiere alto acoplamiento entre el cliente y servidor**.

Luego, en 1996, hace su aparición SOA (Arquitectura orientada a servicios)•**SOA fue descrita por primera vez en el año 1996** (Gartner).

SOA Incentiva a las empresas a **crear servicios interoperables**

Solo **promueve la creación de servicios para lograr la integración de aplicaciones**, por lo que se podría decir que **CORBA es el precursor de SOA**

•Permite que los **desarrollos orientados a servicios sean posibles**.

•SOA como arquitectura no solo trae el concepto de crear servicios sino un **nuevo paradigma que revoluciono la forma en que construimos software de hoy en día**.

**El modelo de programación CORBA**

CORBA es una especificación para crear aplicaciones distribuidas basadas en objetos. La arquitectura y la especificación de CORBA fueron desarrolladas por Object Management Group (OMG). El OMG es un consorcio de varios cientos de proveedores de sistemas de información. El objetivo de CORBA es promover un enfoque orientado a objetos para construir e integrar aplicaciones de software distribuidas.

La especificación CORBA proporciona un modelo amplio y coherente para crear aplicaciones distribuidas al definir:

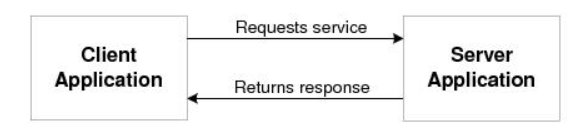
* Un modelo de objetos para crear aplicaciones distribuidas.
* Un conjunto común de objetos de programación de aplicaciones que utilizarán las aplicaciones cliente y servidor.
* Una sintaxis para describir las interfaces de los objetos utilizados en el desarrollo de aplicaciones distribuidas.
* Soporte para el uso de aplicaciones escritas en múltiples lenguajes de programación.

La especificación CORBA describe cómo desarrollar una implementación de CORBA. También describe los enlaces del lenguaje de programación que los desarrolladores usan para desarrollar aplicaciones.

Para ilustrar las ventajas de utilizar la arquitectura CORBA, esta sección compara las primeras técnicas de desarrollo de aplicaciones cliente/servidor con las técnicas de desarrollo CORBA.

**Enfoque anterior a CORBA para el desarrollo de cliente/servidor**

La computación cliente/servidor es una metodología de desarrollo de aplicaciones que permite a los programadores distribuir el procesamiento entre sistemas de máquinas en red, lo que permite un uso más eficiente de los recursos de la máquina. En la informática cliente/servidor, una aplicación consta de dos partes: la aplicación cliente y la aplicación servidor. Estas dos aplicaciones normalmente se ejecutan en diferentes máquinas, conectadas por una red, como se muestra en la siguiente figura.



La aplicación cliente realiza solicitudes de información o servicios y, por lo general, proporciona a los usuarios un medio para mostrar los resultados. La aplicación del servidor cumple con las solicitudes de una o más aplicaciones cliente y, por lo general, realiza funciones de cómputo intensivo.

Las principales ventajas del modelo cliente/servidor son:

* Las funciones informáticas se ejecutan en el sistema de máquina más adecuado.
* Los desarrolladores pueden equilibrar la carga de procesamiento de aplicaciones entre varios servidores.
* Las aplicaciones de servidor se pueden compartir entre numerosas aplicaciones de cliente.

Por ejemplo, los sistemas de escritorio brindan a muchos usuarios comerciales un entorno gráfico fácil de usar para mostrar información. Sin embargo, los sistemas de escritorio pueden tener espacio en disco y memoria restringidos y, por lo general, son sistemas de un solo usuario. Los sistemas de máquinas más grandes y potentes son más adecuados para realizar funciones de cómputo intensivo y proporcionar acceso a múltiples usuarios y acceso a bases de datos compartidas.

Por lo tanto, los sistemas más grandes normalmente ejecutan la parte del servidor de la aplicación. De esta forma, los sistemas de escritorio distribuidos y los servidores en red proporcionan un entorno informático perfecto para implementar aplicaciones cliente/servidor distribuidas.

Aunque el enfoque cliente/servidor no CORBA proporciona los medios para distribuir el procesamiento en una red heterogénea, tiene las siguientes desventajas:

* Para las comunicaciones, la aplicación del cliente debe saber cómo acceder a la aplicación del servidor, incluida cualquier información de protocolo de red necesaria.

Las aplicaciones cliente/servidor pueden usar el mismo protocolo de red único o diferentes protocolos. Si utilizan múltiples protocolos, las aplicaciones deben repetir lógicamente el código específico del protocolo para cada red.

* Las aplicaciones deben manejar conversiones de formato de datos cuando se integran con máquinas que usan diferentes formatos de datos.

Por ejemplo, algunas máquinas leen un valor entero desde la dirección de byte más baja hasta la más alta (little endian), mientras que otras leen la dirección de byte más alta hasta la más baja (big endian). Algunos sistemas de máquinas también pueden usar diferentes formatos para números de punto flotante o cadenas de texto. Si una aplicación envía datos a una máquina que usa un formato de datos diferente, pero la aplicación no convierte los datos, los datos se malinterpretan.

El transporte de datos a través de la red y su conversión a su representación adecuada en el sistema de destino se denomina clasificación de datos . En muchos modelos de cliente/servidor que no son CORBA, las aplicaciones deben realizar todo el cálculo de referencias de datos. La clasificación de datos requiere que la aplicación use características de la red y el sistema operativo para mover datos de una máquina a otra. También requiere que la aplicación realice todas las traducciones de formato de datos para garantizar que los datos se lean de la misma manera que se enviaron.

* Hay menos flexibilidad para la extensión de la aplicación.

El enfoque cliente/servidor no CORBA une las aplicaciones cliente y servidor. Por lo tanto, si cambia la aplicación del cliente o del servidor, el programador debe cambiar la interfaz, la dirección de red y el transporte de red. Además, si el programador transfiere las aplicaciones de cliente y servidor a una máquina que admita una interfaz de red diferente, el programador debe crear una nueva interfaz de red para esas aplicaciones.

**Enfoque CORBA para el desarrollo de cliente/servidor**

El modelo CORBA proporciona un enfoque más flexible para desarrollar aplicaciones distribuidas. El modelo CORBA:

* Separa formalmente las partes de cliente y servidor de la aplicación.

Una aplicación cliente CORBA solo sabe cómo pedir que se haga algo, y una aplicación servidor CORBA solo sabe cómo realizar una tarea que una aplicación cliente le ha pedido que haga. Debido a esta separación, los desarrolladores pueden cambiar la forma en que un servidor realiza una tarea sin afectar la forma en que la aplicación cliente solicita que la aplicación del servidor realice la tarea.

* Separa lógicamente una aplicación en objetos que pueden realizar ciertas tareas, llamadas operaciones.

CORBA se basa en el modelo de computación de objetos distribuidos, que combina los conceptos de computación distribuida (cliente y servidor) y computación orientada a objetos (basada en objetos y operaciones).

En la informática orientada a objetos, los objetos son las entidades que componen la aplicación y las operaciones son las tareas que un servidor puede realizar en esos objetos. Por ejemplo, una aplicación bancaria podría tener objetos para cuentas de clientes y operaciones para depositar, retirar y ver el saldo en las cuentas.

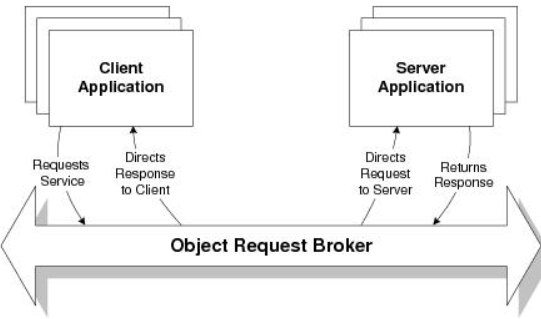
* Proporciona clasificación de datos para enviar y recibir datos con aplicaciones de máquinas locales o remotas

Por ejemplo, el modelo CORBA formatea automáticamente para big o little endian según sea necesario. (Consulte la sección anterior para obtener una descripción de la clasificación de datos).

* Oculta las interfaces de protocolo de red de las aplicaciones.

El modelo CORBA maneja todas las interfaces de red. Las aplicaciones solo ven objetos. Las aplicaciones pueden ejecutarse en diferentes máquinas y, debido a que el ORB maneja todo el código de la interfaz de red, la aplicación no requiere ningún cambio relacionado con la red si luego se implementa en una máquina que admita un protocolo de red diferente.

El modelo CORBA permite que las aplicaciones cliente realicen solicitudes a las aplicaciones del servidor y reciban respuestas de ellas sin conocimiento directo de la fuente de información o su ubicación. En un entorno CORBA, las aplicaciones no necesitan incluir información sobre la red y el sistema operativo para comunicarse; en cambio, las aplicaciones de cliente y servidor se comunican con Object Request Broker (ORB). La siguiente figura muestra el ORB en un entorno cliente/servidor.



CORBA define el ORB como un intermediario entre las aplicaciones cliente y servidor. El ORB entrega las solicitudes de los clientes a las aplicaciones de servidor correspondientes y devuelve las respuestas del servidor a la aplicación de cliente solicitante. Usando un ORB, una aplicación de cliente puede solicitar un servicio sin conocer la ubicación de la aplicación de servidor o cómo la aplicación de servidor cumplirá con la solicitud.

En el modelo CORBA, las aplicaciones cliente solo necesitan saber qué solicitudes pueden realizar y cómo realizarlas; no necesitan estar codificados con ningún detalle de implementación del servidor o de los formatos de datos. Las aplicaciones de servidor solo necesitan saber cómo cumplir con las solicitudes, no cómo devolver datos a la aplicación cliente.

Esto significa que los programadores pueden cambiar la forma en que una aplicación de servidor realiza una tarea sin afectar la forma en que la aplicación cliente solicita que la aplicación de servidor realice esa tarea. Por ejemplo, mientras las interfaces entre las aplicaciones cliente y servidor no cambien, los programadores pueden evolucionar y crear nuevas implementaciones de una aplicación servidor sin cambiar la aplicación cliente; además, pueden crear nuevas aplicaciones cliente sin cambiar las aplicaciones del servidor.

COMPONENTES DE CORBA.

Como se explicó anteriormente, es un estándar para distribuir objetos a través de redes distribuidas.

No está asociado con ningún lenguaje en particular pero puede utilizarse con cualquiera que pueda implementar objetos CORBA. Los objetos se describen en una interface de definición de lenguajes (IDL).

IDL.

Para identificar los servicios que los objetos ofrecen se necesita de un lenguaje independiente y bien definido. La norma ISO/IEC 14750 (ITUT X.920) define dicho lenguaje, al que se conoce como lenguaje de definición de interfaces de ODP, o ODP IDL por su acrónimo en inglés.

En IDL, una interfaz es una descripción de un conjunto de posibles operaciones que un cliente puede solicitar de un objeto. El objeto satisface una interfaz si este puede satisfacer una solicitud de otro objeto. La interface provee mecanismos compuestos que le permiten al objeto soportar varias interfaces.

Repositorio de interfaces.

El repositorio de interfaces (IR) es un servicio que ofrece objetos persistentes que representan la información IDL de las interfaces disponibles en CORBA, de una forma accesible en tiempo de ejecución. Esta información puede ser utilizada por el ORB para realizar peticiones.

La interfaz de invocación dinámica El DII (Dynamic Invocation Interface) es una interfaz que nos permite la construcción dinámica de invocaciones para un determinado objeto.

La invocación es idéntica a la que llega a través de la interfaz estática pero que ya dentro del cliente, logra una flexibilidad en arquitecturas complejas y dinámicas. Una invocación dinámica se compone, de una referencia al objeto, una operación y una lista de parámetros. Todos estos datos se obtienen del Repositorio de Interfaces (IR).

STUB.

Es el intermediario entre el cliente y el ORB. Tomas las llamadas del clientes y las entrega al ORB. Se requiere una clase de stub por cada clase remota. El stub permite que el desarrollador no tenga que trabajar con instrucciones remotas porque son objetos que residen del lado del cliente y representan a los objetos remotos. Aquí se indentifica: Host, puerto e identificador del objeto.

CORBA incluye cuatro componentes:

Skeleton.

Es el intermediario entre ORB y los objetos del servidor . Recibe llamadas del ORB y ejecuta los métodos correspondientes en el servidor sobre el objeto llamado. Cuando el cliente establece un objeto local (con servicio remoto), la petición se realiza por intermedio del protocolo de comunicaciones IIOP a través del ORB. El servidor recibe la petición, busca el objeto definido (compara el esqueleto del método en el módulo esqueleto) lo ejecuta y retorna la respuesta al cliente

ORB (Object Request Broker)

ORB es el componente que gestiona la comunicación de parámetros. De esta manera el manejo e transparente para el cliente que invoca el servicio y el servidor que lo ofrece y soporta cuatro tipos de interfaces de objetos:

• Object Services: Son interfaces para servicios generales. Son usadas en cualquier programa basado en objetos distribuidos.

• Common Facilities: Son interfaces orientadas al usuario final y que se programan por la aplicación específica.

• Domain Interfaces: Son interfaces de dominio específico para las aplicaciones.

• Application Interfaces: Este tipo de interfaz acepta interfaces que no sean estandarizadas y se utilizan en aplicaciones específicas.

Servidor CORBA

El servidor CORBA crea objetos CORBA y los inicializa con un ORB. El servidor coloca las referencias a los objetos CORBA dentro de un servicio de denominación de modo que los clientes puedan acceder a los mismos.

Servicio de nombres

Et servicio es el que mantiene la referencias a los objetos CORBA.

Nodo CORBA Request

Es el que actúa como cliente CORBA.

**Práctica:**

* ¿Qué significan las siglas de CORBA?

Las siglas de **CORBA** significan **Common Object Request Broker Architecture**. Es una especificación desarrollada por el Object Management Group (OMG) para crear aplicaciones distribuidas basadas en objetos​.

* Desarrolle el modelo CORBA y su aplicación

El **modelo CORBA** (Common Object Request Broker Architecture) es una especificación que facilita la creación de aplicaciones distribuidas basadas en objetos. Su propósito es permitir la interoperabilidad entre aplicaciones escritas en diferentes lenguajes de programación y que se ejecutan en diferentes plataformas. CORBA se basa en el concepto de objetos distribuidos, en el cual las aplicaciones cliente y servidor interactúan mediante la invocación de métodos en objetos remotos de manera transparente.

Los componentes clave de CORBA incluyen el **Stub** (del lado del cliente) y el **Skeleton** (del lado del servidor), que gestionan las solicitudes, y un **servicio de nombres** para localizar objetos. CORBA es ampliamente utilizado en sistemas financieros, telecomunicaciones y sistemas empresariales que requieren interoperabilidad y escalabilidad en entornos distribuidos

CORBA es aplicable en entornos donde es necesario integrar aplicaciones distribuidas y heterogéneas. Por ejemplo:

* **Sistemas financieros**: CORBA se usa en aplicaciones bancarias para conectar múltiples sistemas que manejan cuentas, transacciones, y datos financieros en distintos servidores.
* **Telecomunicaciones**: CORBA ha sido ampliamente utilizado en redes de telecomunicaciones, donde es necesario manejar grandes volúmenes de datos distribuidos en varias ubicaciones y plataformas.
* **Sistemas empresariales**: En grandes corporaciones, CORBA facilita la interoperabilidad entre sistemas ERP, CRM y otros sistemas de gestión empresarial desarrollados por diferentes proveedores y en diferentes lenguajes.

**SOA**

La arquitectura orientada a servicios (SOA) es un paradigma de diseño de software en el que los componentes de la aplicación brindan servicios a otros componentes a través de un protocolo de comunicación de red. Sus principios no se ven afectados por los proveedores u otras tecnologías. Varios servicios se comunican entre sí en una arquitectura orientada a servicios de dos maneras: pasando datos o mediante dos o más servicios coordinando una acción. Este es solo un ejemplo de una definición de Arquitectura Orientada a Servicios. Wikipedia tiene un artículo mucho más detallado sobre el tema.

**Características de SOA**

Si bien las nociones básicas de la Arquitectura Orientada a Servicios difieren de una organización a otra, existen seis principios clave que sustentan todo el concepto. - El valor empresarial es uno de estos valores básicos.

- Objetivos estratégicos.

- La interoperabilidad inherente al sistema.

- Servicios que se comparten.

- Adaptabilidad.

- Refinamiento por evolución.

Desde la computación distribuida de formato más antiguo hasta la arquitectura orientada a servicios hasta la computación en la nube, cada uno de estos conceptos esenciales se puede ver en un continuo (algo que a menudo se ve como una rama de la arquitectura orientada a servicios).

**Patrones de SOA**

Cada elemento de construcción de la Arquitectura Orientada a Servicios tiene tres roles: proveedor de servicios, intermediario de servicios, registro de servicios, depósito de servicios y solicitante/consumidor de servicios.

El proveedor de servicios colabora con el registro de servicios para analizar los motivos y procedimientos de los servicios que se entregan, incluida la seguridad, la disponibilidad, los precios y otros factores. Esta posición también define la categoría del servicio y si es necesario algún acuerdo comercial.

El corredor de servicios proporciona información sobre el servicio a las personas que lo desean. El alcance del intermediario lo determina la persona que lo instala.

El solicitante del servicio busca entradas en el registro del corredor y luego las conecta con el proveedor del servicio. Es posible que puedan acceder o no a numerosos servicios, según las capacidades del solicitante del servicio.

**Implementando SOA**

Existe una variedad de tecnologías que se pueden utilizar para desarrollar una arquitectura orientada a servicios (SOA), según cuál sea su objetivo final y lo que intente lograr.

Los servicios web se utilizan comúnmente para desarrollar arquitectura orientada a servicios, lo que hace que los "bloques de construcción funcionales sean accesibles a través de los protocolos de Internet convencionales".

SOAP, que significa Protocolo simple de acceso a objetos, es un ejemplo de un estándar de servicio web. SOAP "es una especificación de protocolo de mensajes para transferir información estructurada en el desarrollo de servicios web en redes informáticas", para decirlo de manera sucinta. Aunque SOAP inicialmente se recibió con escepticismo, ha crecido en popularidad y aceptación desde su creación en 2003. Jini, COBRA y REST son algunas otras opciones para construir Arquitectura Orientada a Servicios.Cada elemento de construcción de la Arquitectura Orientada a Servicios tiene tres roles: proveedor de servicios, intermediario de servicios, registro de servicios, depósito de servicios y solicitante/consumidor de servicios.

Vale la pena señalar que los diseños pueden "funcionar independientemente de una tecnología específica", lo que implica que se pueden implementar de varias maneras, incluidos los sistemas de mensajería como ActiveMQ, Apache Thrift y SORCER.

**Importancia de SOA**

La arquitectura orientada a servicios tiene numerosas ventajas, particularmente en una organización basada en servicios web.

Para generar código reutilizable, utilice la arquitectura orientada a servicios: esto no solo ahorra tiempo durante el proceso de desarrollo, sino que también elimina la necesidad de reinventar la rueda del código cada vez que se requiere un nuevo servicio o proceso. Debido a que todo se ejecuta a través de una interfaz común, la Arquitectura Orientada a Servicios también permite el uso de múltiples lenguajes de codificación.

Utilice la arquitectura orientada a servicios para fomentar la interacción: la arquitectura orientada a servicios establece un método estándar de comunicación, lo que permite que sistemas y plataformas dispares operen de forma independiente entre sí. La arquitectura orientada a servicios también puede funcionar a través de firewalls debido a esta interacción, lo que permite que "las empresas compartan servicios que son críticos para las operaciones".

La escalabilidad se puede lograr mediante el uso de una arquitectura orientada a servicios (SOA): la capacidad de escalar una empresa para satisfacer las necesidades del cliente es fundamental, pero ciertas dependencias pueden obstruir esa escalabilidad. El contacto con el servicio al cliente se reduce al usar la arquitectura orientada a servicios, lo que permite una mayor escalabilidad.

**Práctica:**

* Cuáles son las características de SOA?

- El valor empresarial es uno de estos valores básicos.

- Objetivos estratégicos.

- La interoperabilidad inherente al sistema.

- Servicios que se comparten.

- Adaptabilidad.

- Refinamiento por evolución.

* Cuál es la importancia de utilizar SOA?
* SOA permite crear servicios reutilizables, lo que reduce la duplicación de esfuerzos y acelera el desarrollo.
* Facilita la interacción entre diferentes sistemas y tecnologías, permitiendo la integración de sistemas heterogéneos.
* SOA permite a las organizaciones escalar sus servicios para satisfacer la demanda sin comprometer la funcionalidad.
* Establece un método estándar para la comunicación entre servicios, facilitando la colaboración entre diferentes plataformas.
* Reduce la necesidad de desarrollar nuevas soluciones para cada aplicación, optimizando el uso de recursos.