UNIVERSIDAD

DE LA

SIERRA SUR

INGENIERÍA DE SOFTWARE I

MODELOS DE PROCESOS DE SOFTWARE

Profesor: Rolando Pedro Gabriel

Grupo 506 B

Noviembre 2021

Índice de contenido

Índice de figuras

Modelos de procesos de software

Modelo cascada

El modelo cascada es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar ala finalización de la etapa anterior. El modelo de la cascada es el paradigma más antiguo de la ingeniería de software.

El modelo de la cascada, a veces llamado ciclo de vida clásico, sugiere un enfoque sistemático y secuencial para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue, para concluir con el apoyo del software terminado. Sirve como un modelo de proceso útil en situaciones en las que los requerimientos son fijos y el trabajo avanza en forma lineal hacia el final.

Figura 1: Modelo cascada

Variante del modelo cascada

Una variante de la representación del modelo de la cascada se denomina modelo en V, donde se aprecia la relación entre las acciones para el aseguramiento de la calidad y aquellas asociadas con la comunicación, modelado y construcción temprana. A medida que el equipo de software avanza hacia abajo desde el lado izquierdo de la V, los requerimientos básicos del problema mejoran hacia representaciones técnicas cada vez más detalladas del problema y de su solución. Una vez que se ha generado el código, el equipo sube por el lado derecho de la V, y en esencia ejecuta una serie de pruebas (acciones para asegurar la calidad) que validan cada uno de los modelos creados cuando el equipo fue hacia abajo por el lado izquierdo.

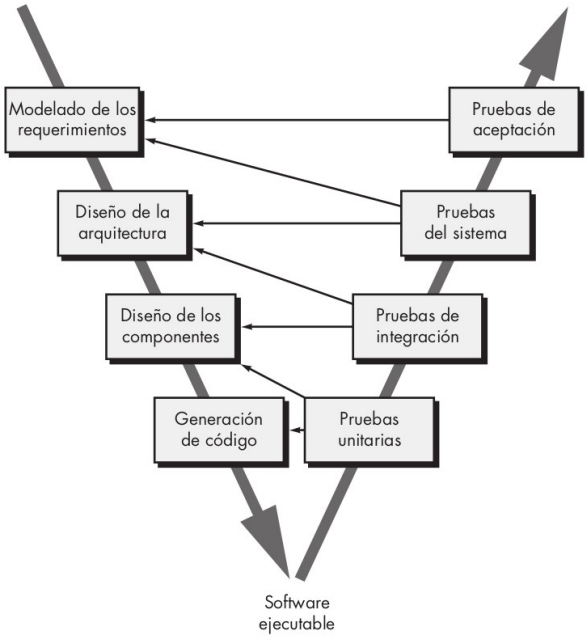

Figura 2: Modelo en V

Principales etapas

1. **Análisis de los requisitos del software:** el proceso de recopilación de los requisitos se centra e intensifica especialmente en el software. El ingeniero de software debe comprender el ámbito de la información del software así como la función, el rendimiento y las interfaces requeridas.
2. **Ingeniería y Análisis del Sistema:** Debido a que el software es siempre parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software.
3. **Diseño:** el diseño del software se enfoca en cuatro atributos distintos del programa; la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación.
4. **Codificación:** el diseño debe traducirse en una forma legible para la maquina. Si el diseño se realiza de una manera detallada, la codificación puede realizarse mecánicamente.
5. **Prueba:** una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.
6. **Mantenimiento:** el software sufrirá cambios después de que se entrega al cliente.

Inconvenientes

Entre los problemas que en ocasiones surgen al aplicar el modelo de la cascada se encuentran los siguientes:

- Aunque el modelo lineal acepta repeticiones, lo hace en forma indirecta. Como resultado, los cambios generan confusión conforme el equipo del proyecto avanza.
- Es difícil para el cliente enunciar en forma explícita todos los requerimientos. El modelo de la cascada necesita que se haga y tiene dificultades para aceptar la incertidumbre natural que existe al principio de muchos proyectos.
- El cliente debe tener paciencia. No se dispondrá de una versión funcional del(de los programa(s) hasta que el proyecto esté muy avanzado. Un error grande sería desastroso

si se detectara hasta revisar el programa en funcionamiento.

Bradac encontró que la naturaleza lineal del ciclo de vida clásico llega a “estados de bloqueo” en los que ciertos miembros del equipo de proyecto deben esperar a otros a fin de terminar tareas interdependientes. En realidad, ¡el tiempo de espera llega a superar al dedicado al trabajo productivo! Los estados de bloqueo tienden a ocurrir más al principio y al final de un proceso secuencial lineal.

Ventaja
La documentación se produce en cada fase y este cuadra con otros modelos del proceso de ingeniería.
Desventaja
inflexibilidad al dividir el proyecto en distintas etapas.

Modelos de Proceso Evolutivo

Los modelos evolutivos son de tipo iterativo, se caracteriza por la manera en la que permiten desarrollar versiones cada ves mas completas del software. Tiene su base en la idea del desarrollo de una implementación inicial, exponiéndola a comentarios de los usuarios y refinando por medio de muchas versiones hasta conseguir un modelo adecuado. Los modelos mas comunes de procesos evolutivos el modelo de prototipos y el modelo evolutivo, aparte hay dos tipos de desarrollo evolutivo:

1. El proceso no visible: Los administradores hacen entregas regulares para medir el progreso
2. Los sistemas tienen una estructura deficiente: Los cambios continuos corrompen la estructura de software.

Los participantes ven lo que parece ser una versión funcional del software, pero no se consideró la calidad, la facilidad de mantenimiento, por la prisa. Los usuarios exigen el prototipo como producto funcional

Desarrollo de proceso prototipos

El prototipo debe ser construido en poco tiempo, usando los programas adecuados y no se debe utilizar muchos recursos. El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente para una retroalimentación; gracias a esta se refinan los requisitos del software que se desarrollará. La interacción ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.

Ventajas

- Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
- También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina
- Se puede reutilizar el código.

Desventajas

- El usuario tiende a crearse unas expectativas cuando ve el prototipo de cara al sistema final. A causa de la intención de crear un prototipo de forma rápida, se suelen desatender aspectos importantes, tales como la calidad y el mantenimiento a largo plazo, lo que obliga en la mayor parte de los casos a reconstruirlo una vez que el prototipo ha cumplido su función. Es frecuente que el usuario se muestre reacio a ello y pida que sobre ese prototipo se construya el sistema final, lo que lo convertiría en un prototipo evolutivo, pero partiendo de un estado poco recomendado.
- En aras de desarrollar rápidamente el prototipo, el desarrollador suele tomar algunas decisiones de implementación poco convenientes (por ejemplo, elegir un lenguaje de programación incorrecto porque proporcione un desarrollo más rápido). Con el paso del tiempo, el desarrollador puede olvidarse de la razón que le llevó a tomar tales decisiones, con lo que se corre el riesgo de que dichas elecciones pasen a formar parte del sistema final.

Desarrollo de proceso espiral

El modelo en espiral, se basa en hacer prototipos con aspectos controlados, sistemáticos del modelo de cascada, se puede hacer un desarrollo rápido de versiones cada vez mas complejas, es un generador del modelo del proceso cíclico para el crecimiento incremental de un sistema y su implementación, disminuyendo el rango de riesgo. El conjunto de puntos de referencia de anclaje puntual asegura el compromiso del participante con soluciones factibles.

1. Un ciclo en espiral empieza con la elaboración de los objetivos, como el rendimiento y la funcionalidad. Se enumeran formas alternativas de alcanzar estos objetivos y sus restricciones.
2. Cada alternativa se evalúa contra cada objetivo y se identifican las fuentes de riesgo.
3. El siguiente paso es resolver el riesgo mediante actividades como detallar más el análisis, la construcción de prototipos y la simulación.
4. Una vez que se han analizado los riesgos se lleva a cabo cierto desarrollo, seguido de una actividad de planificación para la siguiente fase.

Datos interesantes

- Propuesto en primer lugar por Barry Boehm.
- Es un modelo con la naturaleza iterativa de hacer
- Prototipos y los aspectos controlados y sistémicos del modelo de cascada.
- Representa el proceso de desarrollo de software como una espira

Definición de objetivos

- Definen los objetivos específicos.
- Identifica las restricciones del proceso y el producto.
- Se traza un plan detallado de gestión.
- Se identifican los riesgos del proyecto. Dependiendo de los riesgos se planean las estrategias.

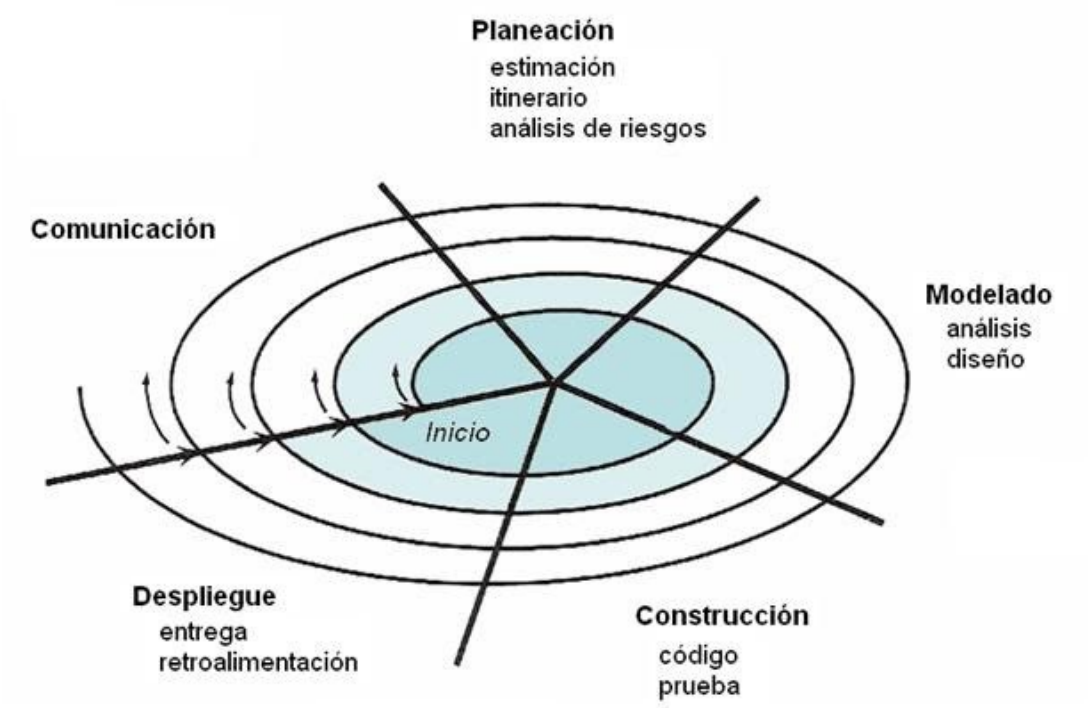

Figura 3: Modelo espiral

Desarrollo de proceso concurrente

El Modelo de Desarrollo Concurrente conocido además como Ingeniería Concurrente dado por Davis Sitaram, se puede representar en forma de esquema como una serie de actividades técnicas importantes, tareas y estados asociados a ellas. Provee una meta-descripción del proceso del software. El modelo concurrente tiene la capacidad de describir las múltiples actividades del software ocurriendo simultáneamente.

La concurrencia se logra de dos formas:

1. Las actividades de sistemas y de componentes ocurren simultáneamente y pueden modelarse con el enfoque orientado a objetos.
2. Una aplicación cliente/servidor típica se implementa con muchos componentes, cada uno de los cuales se pueden diseñar y realizar concurrentemente.

Ventajas

- Excelente para proyectos en los que se conforman grupos de trabajo independientes.
- Proporciona una imagen exacta del estado actual de un proyecto.

Desventajas

- Si no se dan las condiciones señaladas no es aplicable.
- Si no existen grupos de trabajo no se puede trabajar en este método.

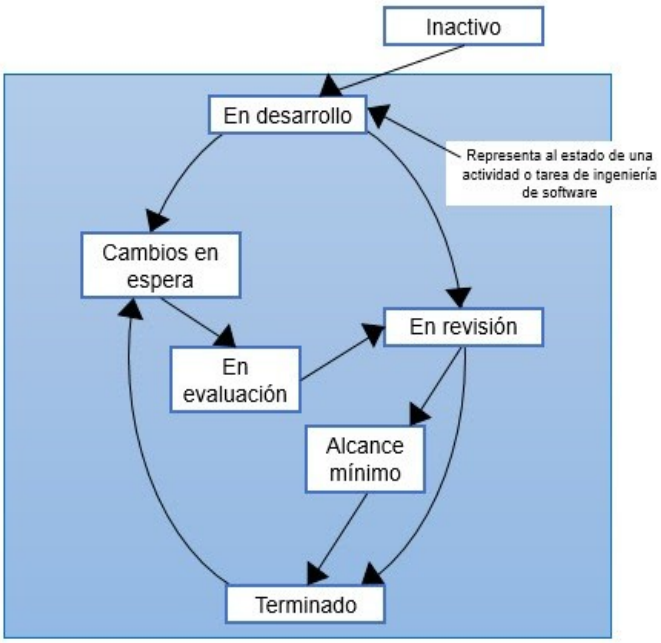

Figura 5: Modelo proceso concurrente

Modelos de proceso incrementales

Fue propuesto por Harlan Mills en el año 1980, como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema.
Este modelo se conoce también como:

- Método de las comparaciones limitadas sucesivas.
- Ciencia de salir del paso
- Método de atacar el problema por ramas

El modelo incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Cuando se utiliza un modelo incremental, el primer incremento es a menudo un producto esencial, sólo con los requisitos básicos. Este modelo se centra en la entrega de un producto operativo con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

Ventajas

Entre las ventajas que puede proporcionar un modelo de este tipo encontramos las siguientes:

- Mediante este modelo se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software.
- Es un modelo más flexible, por lo que se reduce el coste en el cambio de alcance y requisitos.
- Es más fácil probar y depurar en una iteración más pequeña.
- Es más fácil gestionar riesgos.
- Cada iteración es un hito gestionado fácilmente .

Inconvenientes

Para el uso de este modelo se requiere una experiencia importante para definir los incrementos y distribuir en ellos las tareas de forma proporcionada. Entre los inconvenientes que aparecen en el uso de este modelo podemos destacar los siguientes:

- Cada fase de una iteración es rígida y no se superponen con otras.
- Pueden surgir problemas referidos a la arquitectura del sistema porque no todos los requisitos se han reunido, ya que se supone que todos ellos se han definido al inicio

- Difícil de aplicar a sistemas transaccionales que tienden a ser integrados y a operar como un todo.
- Riesgos largos y complejos.
- Pueden aumentar el coste debido a las pruebas.
- Los errores en los requisitos se detectan tarde.

Métodos Formales

El modelo de métodos formales agrupa actividades que llevan a ala especificación matemática formal del software de computo. Los métodos formales permiten especificar, desarrollar y verificar un sistema basado en una computadora por medio del empleo de una notación matemática rigurosa. Cuando durante el desarrollo se usan métodos formales, se obtiene un mecanismo para eliminar mucho los problemas difíciles de vencer con otros paradigmas de la ingeniería de software.
El modelo de los métodos formales no es el más seguido, promete un software libre de defectos. Un método formal es una técnica basada en matemáticas, usada para describir sistemas de hardware o software, nos permiten representar la especificación del software, verificación y diseño de componentes mediante notaciones matemáticas.
Su uso permite plantear de manera clara la especificación de un sistema, generando modelos que definen el comportamiento en términos del “qué debe hacer” y no del “cómo lo hace”.

Niveles de los métodos formales

- **Especificación formal:** Es la descripción de un sistema que utiliza notaciones matemáticas para describir las propiedades que dicho sistema debe tener, sin preocuparse por la forma en que se obtienen las propiedades. (Describe lo que el sistema debe hacer sin decir como se va a hacer).
- **Método axiomático o algebraico:** Se establece el significado de las operaciones a traves de relaciones entre operaciones (axiomas).
- **Método constructivo u operacional:** Se define cada operación por sí misma, independientemente de las otras.
- **Verificación formal:** Método de validación estática (validación a través del código del programa) en el que partiendo de un conjunto axiomático, reglas de inferencia y un lenguaje lógico, se puede encontrar una demostración o prueba de corrección del algún programa.
- **Verificación estática completa:** Se lleva a cabo a mano o en ocasiones se hace uso de herramientas tales como probadores de teoremas.
- **Verificación estática parcial:** Se hace uso de entornos capaces de detectar algunas propiedades relativas a la corrección de un programa dado, aunque sin llegar a probar la corrección total del software.
- **Verificación dinámica:** Se basa en que las aserciones insertadas en el código pueden mediante un compilador ser transformadas en código ejecutable.

- **Demostración automática de teoremas:** Se encarga de la demostración de teoremas matemáticos mediante programas de ordenador.

Preocupaciones acerca de su aplicabilidad en un ambiente de negocios

- El desarrollo de los modelos formales consume mucho tiempo y es caro a comparación con otros.
- Debido a que pocos desarrolladores de software tienen la formación necesaria para aplicar métodos formales, se requiere de mucha capacitación.
- Es difícil utilizar los modelos como mecanismo de comunicación para los clientes sin complejidad técnica.
- El enfoque de los métodos formales ha ganado partidarios entre los desarrolladores que deben construir software de primera calidad en seguridad por lo cual es usado para el control de aeronaves, equipos médicos, etc.

Los métodos formales surgieron como puntos de vista analíticos con los que es posible verificar el desarrollo de sistemas mediante la lógica y las matemáticas, lo que aporta grandes ventajas para mejorar la calidad de los programas y por tanto la Ingeniería de Software. En este campo del conocimiento, la especificación formal es una de las más importantes fases del ciclo de vida, labor que requiere mucho cuidado ya que su función es garantizar que tanto el funcionamiento como el desempeño del programa sean correctos, bajo cualquier situación.

Usos

En la Ingeniería de Software los métodos formales se utilizan para:

- Las políticas de los requisitos. políticas de seguridad formal , como confidencialidad o integridad de datos.
- La especificación. descripción matemática basada en el comportamiento del sistema, que utiliza tablas de estado o lógica matemática.
- Las pruebas de correspondencia entre la especificación y los requisitos. Es necesario demostrar que el sistema, tal como se describe en la especificación, establece y conserva las propiedades de las políticas de los requisitos.
- Las pruebas de correspondencia entre el código fuente y la especificación. técnicas formales que se crearon inicialmente para probar la correctitud del código, pero pocas veces se logra debido al tiempo y costo implicados, pero pueden aplicarse a los componentes críticos del sistema.
- Pruebas de correspondencia entre el código máquina y el código fuente. Este tipo de pruebas raramente se aplica debido al costo, y a la alta confiabilidad de los compiladores modernos.

Modelo basado en componentes reutilizables

Sucede cuando las personas que trabajan en un proyecto encuentran y conocen diseños o códigos similares al que van a ocupar, estos códigos los estudian, modifican e implementan en su respectivo sistema, esta reutilización es a menudo es indispensable para la realización y el desarrollo rápido de un sistema. Este enfoque basado en reutilización de un sistema se compone en gran medida de componentes de software reutilizables y de algunos macros de trabajo de integración para estos.

El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo en espiral. Es evolutivo por naturaleza y exige un enfoque iterativo para la creación del software. Las etapas intermedias orientado a la reutilización son diferentes, y estas son:

- **El análisis de componentes:** buscan los componentes para implementar esta especificación.
- **Modificación de requerimientos:** se analizan usando información acerca de los componentes que se han descubierto, se modifican para reflejar los componentes disponibles.
- **Diseño del sistema de reutilización:** los diseñadores tiene en cuenta los componentes que se reutilizan y organizan el marco de trabajo.
- **Desarrollo e integración:** la integración del sistema parte del proceso de desarrollo.

Entre sus ventajas contiene que se reutiliza el software, simplifica las pruebas (permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados), simplifica el mantenimiento del sistema(cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema), mayor calidad (dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo).
La idea general es reutilizar el trabajo de otras personas para tus fines, y de esa forma agilizar la creación y realización de los proyectos que tengas que hacer, pero igual es de importante entender que se esta reutilizando con el fin de poder aplicar esos conocimientos a futuro y no ser tan dependiente de este modelo de trabajo, aunque siempre cabe la posibilidad de volver a este modelo.

Proceso Unificado

Es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.
Al principio de la década de 1990, James Rumbaugh, Grady Booch e Ivar Jacobson, comenzaron a trabajar en un “método unificado” que intentaría obtener los mejores rasgos y características de los modelos tradicionales del proceso de software, pero de forma que implementara muchos de los mejores principios de desarrollo ágil de software.

El resultado fue UML (Lenguaje de modelado unificado), que contiene una notación robusta para el modelado y desarrollo de sistemas orientados a objetos.

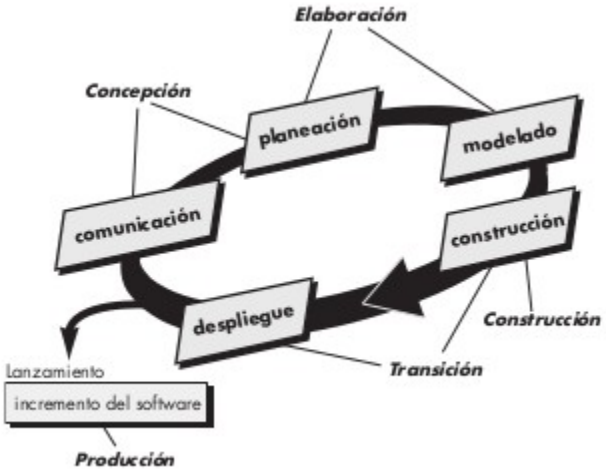

Figura 6: Modelo Proceso Unificado

Características:

• Reconoce la importancia de la comunicación con el cliente y los métodos para recibir su punto de vistas respecto al sistema.

• Hace énfasis en la importancia de la arquitectura de software: que sea comprensible, que pueda modificarse a futuro y que se reutiliza.

• Sugiere un flujo de proceso iterativo e incremental.

Fases

Fase de concepción: Se identifican requerimientos del negocio, se propone una arquitectura y se desarrolla un plan para la naturaleza iterativa e incremental del producto.

Fase de elaboración: Incluye actividades de comunicación y modelado del modelo general del proceso. La elaboración amplía los casos de uso preliminares desarrollados como parte de la fase de concepción y aumenta la representación y aumenta la representación de la arquitectura para incluir cinco puntos de vista distintos del software: los modelos de casos de uso, de requerimientos, del diseño, de la implementación y del despliegue.
Al finalizar la fase se revisa con cuidado el plan al fin de asegurar que el alcance , riesgos y fechas de entrega siguen siendo razonables.

Fase de construcción: Se desarrolla o adquiere los componentes del software que harán que cada caso de uso sea operativo para los usuarios finales.
Para lograrlo, se completan los modelos de requerimientos y diseño que se comenzaron en la fase de elaboración a fin de que reflejen la versión de incremento final del incremento de software. Después se implementan en código fuente todas las características y funciones necesarias para el incremento de software. A medida que se implementan componentes, se diseñan y efectúan pruebas unitarias para cada uno. Además se realizan actividades de integración. Se emplean casos de uso para obtener un grupo de pruebas de aceptación que se ejecuten antes de comenzar la siguiente fase.

Fase de transición: Incluye las últimas etapas de la actividad general de construcción y la primera parte de la actividad de despliegue general (entrega u retroalimentación). El software llega al usuario final para pruebas beta, son estos quienes reportan los defectos así como los cambios necesarios.
Además, el equipo de software general genera la información de apoyo necesaria(por ejemplo manuales de usuario, guías de solución de problemas, procedimientos de instalación, etc.) que se requiere para el lanzamiento.
Al finalizar la fase de transición, el software incrementado se convierte en el producto utilizable que se lanza.

Fase de producción: Actividad de despliegue del proceso general. En esta fase se vigila el uso que se da al software, se brinda apoyo para el ambiente de operación (infraestructura) y se reportan defectos y solicitudes de cambio para su evaluación.

Modelos en PU

1. **Modelo del negocio:** establece una abstracción de la organización.
2. **Modelo del dominio:** establece el contexto del sistema.
3. **Modelo de casos de uso:** establece los requisitos funcionales del sistema.
4. **Modelo de análisis :** establece un diseño de las ideas.
5. **Modelo de diseño:** establece el vocabulario del problema y su solución.
6. **Modelo del proceso :** establece los mecanismos de concurrencia y sincronización del sistema.
7. **Modelo de despliegue:** establece la tipología hardware sobre la cual se ejecutará el sistema.
8. **Modelo de implementación:** establece las partes que se utilizarán para ensamblar y hacer disponible el sistema físico.
9. **Modelo de pruebas:** establece las formas de validar y verificar el sistema.

Scrum

Scrum es un método de desarrollo ágil de software concebido por Jeff Sutherland y su equipo de desarrollo a principios de la década de 1990. En años recientes, Schwaber y Beedle [Sch01a] han desarrollado más los métodos Scrum.

Los principios Scrum son congruentes con el manifiesto ágil y se utilizan para guiar actividades de desarrollo dentro de un proceso de análisis que incorpora las siguientes actividades estructurales: requerimientos, análisis, diseño, evolución y entrega. Dentro de cada actividad estructural, las tareas del trabajo ocurren con un patrón del proceso llamado sprint. El trabajo realizado dentro de un sprint se adapta al problema en cuestión y se define y con frecuencia se modifica en tiempo real por parte del equipo Scrum. El flujo del proceso Scrum se muestra en la figura:

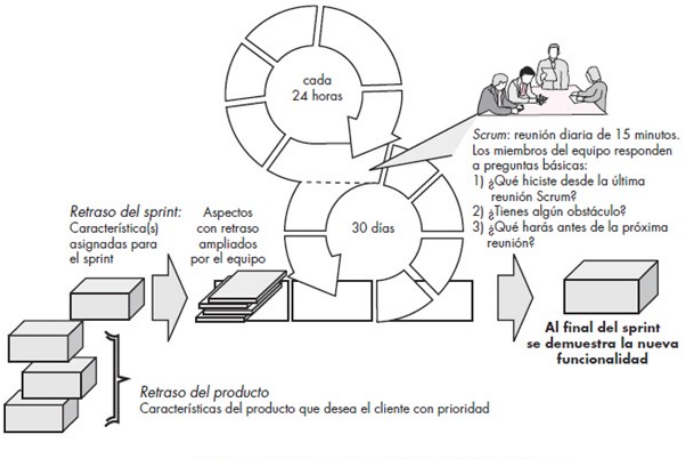

Figura 7: SCRUM

Scrum acentúa el uso de un conjunto de patrones de proceso del software que han demostrado ser eficaces para proyectos con plazos de entrega muy apretados, requerimientos cambiantes y negocios críticos. Cada uno de estos patrones de proceso define un grupo de acciones de desarrollo:

Retraso: lista de prioridades de los requerimientos o características del proyecto que dan al cliente un valor del negocio. El gerente del proyecto evalúa el retraso y actualiza las prioridades según se requiera.

Sprints: consiste en unidades de trabajo que se necesitan para alcanzar un requerimiento definido en el retraso que debe ajustarse en una caja de tiempo predefinida. Durante el sprint no se introducen cambios. Así, el sprint permite a los miembros del equipo trabajar en un ambiente de corto plazo
pero estable.

Reuniones Scrum: son reuniones breves (de 15 minutos, por lo general) que el equipo
Scrum efectúa a diario. Hay tres preguntas clave que se pide que respondan todos los miembros del equipo:

- ¿Qué hiciste desde la última reunión del equipo?
- ¿Qué obstáculos estás encontrando?

- ¿Qué planeas hacer mientras llega la siguiente reunión del equipo

Demostraciones preliminares: entregar el incremento de software al cliente de modo que la funcionalidad que se haya implementado pueda demostrarse al cliente y éste pueda evaluarla. Es importante notar que las demostraciones preliminares no contienen toda la funcionalidad planeada, sino que éstas se entregarán dentro de la caja de tiempo establecida.

Ventajas de Scrum

Scrum representa una de las mejores metodologías ágiles para el trabajo en equipo, de modo que listar sus beneficios también exhibe su potencial:

- Ayuda a establecer metas cuantificables y, así, mantener al equipo ocupado todo el tiempo para garantizar la productividad.
- La flexibilidad del método Scrum te permite modificar el hilo conductor de los hechos en cualquier momento, lo que favorece la resolución de conflictos u obstáculos sobre la marcha.
- Muestra el progreso del sprint o iteración de requerimientos.
- Evita el perfeccionismo innecesario.
- Se obtienen resultados rápidos y periodos de prueba muy cortos. Así, un producto puede estar listo para venta al público en poco tiempo.
- El método ofrece una mirada cercana y transparente del proyecto a todas las partes interesadas. Esto aporta confianza a los clientes.
- Motiva al cierre de los sprints (mejor tasa de cumplimiento).
- Te muestra una visión completa del proyecto.
- Usando un Software para Scrum, resulta muy fácil priorizar constantemente y saber distinguir entre los pendientes que pueden esperar y las tareas cruciales.

Desventajas de Scrum

- Nada puede ser perfecto. Por eso, tenemos que hablar de las desventajas de Scrum
- Al momento de trabajar con Scrum, todo el equipo debe conocer a fondo sus principios y marco teórico. Tiene que haber conocimiento de los roles de Scrum o podría haber dispersión y choque de funciones.
- En gran parte, el éxito de un proyecto conducido con el método Scrum dependerá de la preparación y experiencia del Scrum Master (más que de la innovación, creatividad o calidad de los insumos).
- Si por algún motivo quedan tareas sin finalizar, el resto de pendientes comenzará a postergarse indefinidamente, pues a menudo hay una relación lógica y secuencial entre las actividades del tablero.
- Puede ser difícil el manejo de Scrum en proyectos complejos.

Metodología Kanban

La metodología Kanban es un sistema de producción tan eficiente como efectivo. Forma parte de las metodologías ágiles y su objetivo es gestionar la realización de las tareas hasta su finalización.

¿Cómo funciona la metodología Kanban?

Kanban es una palabra japonesa formada por Kan, que quiere decir visual, y Ban, que significa tarjeta. Por lo tanto, Kanban hace referencia a las tarjetas visuales.

Esta metodología es muy sencilla, se puede actualizar y los equipos de trabajo la pueden asumir sin problema. Al ser un método visual permite que con un simple vistazo se conozca el estado de los proyectos y se puedan asignar nuevas tareas de manera muy efectiva. Para aplicarlo, es necesario un tablero de tareas con el que poder mejorar el trabajo y tener un ritmo sostenible.

Ventajas

A través de los principios en los que se basa este método ya podemos intuir las numerosas ventajas que conlleva su aplicación en cualquier empresa. Estas son los 4 beneficios indiscutibles.

1. **Transparencia:** Los tiempos de entrega son más cortos y hay una mayor fiabilidad en los mismos. Todo el mundo sabe cuál es su tarea y en qué momento está de su ciclo.
2. **Evita tareas ineficientes:** Se evita la sobreproducción y la limitación de los recursos, lo que supone contar con una mayor disponibilidad de materiales.
3. **Control de las tareas:** El tiempo de producción es más rápido, por tanto, se reduce el control del esfuerzo y se mejora la planificación. Esto afecta directamente a la mayor productividad en el área de compras, abastecimiento y control. Aumenta la rotación de los inventarios y se necesita una menor capacidad de almacenamiento.
4. **Flexibilidad:**Como todo el equipo sabe perfectamente cuál es su tarea y la realiza con eficacia, si surge alguna tarea imprevista existe una capacidad de respuesta que permite atenderla.

En definitiva, la metodología Kanban está indicada para las empresas que necesiten cierta flexibilidad a la hora de manejar nuevas entradas de tareas y poder realizar un buen seguimiento de las mismas. Además, permite priorizar, realizar informes precisos y supervisar adecuadamente el trabajo en equipo.

Desventajas

1. Coste, si se usa este método para unidades muy grandes, el almacenamiento del sistema de Kanban será muy costoso.
2. Es limitado, al limitar el número de tareas, cuando se trata de trabajos inmensos no es posible aplicar esta metodología ya que muchos de los trabajadores quedarían desocupados.

3. Kanban no es óptimo para todo tipo de proyectos. Kanban asume sistemas de producción repetitivos, es por eso que las variaciones o los eventos inesperados pueden afectar negativamente al resultado fina.
4. No permite anticiparse a grandes aumentos de la demanda. Con Kanban resulta difícil manejar cambios de gestión provocados por la acumulación de nuevas tareas, lo que podría provocar un desbordamiento de trabajo.
5. No es una técnica específica del desarrollo software.
6. El sistema no tiene ninguna anticipación en caso de fluctuaciones muy grandes e imprevisibles en la demanda. Puede anticiparse a ellas, pero no solucionarlas.
7. Es aplicable a producciones de tipo «masa» para las cuales el número de referencias no es muy elevado, y la petición es regular o a reducidas variaciones.
8. Reducir el número de Kanban sin aportar de mejoramientos radicales al sistema de producción, arrastrará retrasos de entrega y de espera entre operaciones y en consecuencia, pérdidas importantes.
9. No ha tenido el éxito ni ha llegado al óptimo funcionamiento cuando ha sido implementado en organizaciones occidentales. Uno de las principales causas de ello, las enormes diferencias culturales.
10. Sólo funcionará bien en el contexto de un sistema justo a tiempo en general, y de la característica JIT (Just in time) de reducción del tiempo de preparación y del tamaño del lote.
11. Únicamente se debe aplicar Kanban a las partes que se consumen el mismo día de su producción.
12. Al usarlo en unidades muy costosas y/o muy grandes, el almacenamiento y manejo son muy costos.

Fases de la Metodología Kanban

1. **Instrucción de todo el personal acerca de la metodología Kanban:** Esta fase ayuda a cada miembro del equipo a tener los conocimientos y conciencia en cuanto al manejo de la metodología y los beneficios de su aplicación.
2. **Implementación del sistema Kanban en los componentes con más problemas:** Esta implementación se lleva a cabo en los componentes que presenten mayores problemas o dificultades, todo con la intención de facilitar su ejecución o manufactura. La segunda fase permite, a su vez, enfatizar o resaltar problemas que no se habían detectado.
3. I**mplementación de Kanban en los componentes restantes:** Solo se implementa el sistema en el resto de los componentes cuando han sido halladas soluciones para aquellos que presenten problemas de mayor envergadura. En esta fase, los miembros del equipo tienen una mayor conciencia y manejo de Kanban, por lo tanto, ya conocen sus ventajas. Además, se supone que los operarios de la empresa, categorizados por áreas, ya manejan pormenorizadamente el

sistema Kanban, de modo que es fundamental informarles cuándo se estará trabajando en su área y escuchar sus dudas / opiniones.

4. **Revisión del sistema o metodología Kanban:** Consiste en una revisión exhaustiva del sistema para determinar qué puntos deben reordenarse. En esta fase, es muy importante constatar que ningún trabajo se realice fuera de secuencia y que cualquier problema se notifique lo más pronto posible al supervisor.

Cristal

Crystal es una metodología de desarrollo de Software ágil, que en realidad está considerada como una familia de metodologías debido a que se subdivide en varios tipos de metodologías en función a la cantidad de personas que vayan a conformar el proyecto.
Alistar Cockburn y Jim Highsmith crearon la familia Cristal con el fin de lograr un enfoque de desarrollo de software que coloca un premio en la "manejabilidad" durante lo que Cockburn caracteriza como un "juego colaborativo de inventiva y comunicación con recursos limitados, con una meta primaria consistente en la entrega de software útil y en funcionamiento y una meta secundaria de prepararse para el juego siguiente".

El nombre Cristal deriva de la caracterización de los proyectos según 2 dimensiones, tamaño y complejidad. Por ejemplo:
Clear es para equipos de hasta 6 personas o menos.
Amarillo para equipos entre 7 a 20 personas.
Naranja para equipos entre 21 a 40 personas.
Roja para equipos entre 41 a 80 personas.
Marron para equipos entre 81 a 200 personas.
La familia Cristal dispone un código de color para marcar la complejidad de una metodología: cuanto más oscuro un color, más "pesado" es el método. Cuanto más crítico es un sistema, más rigor se requiere.

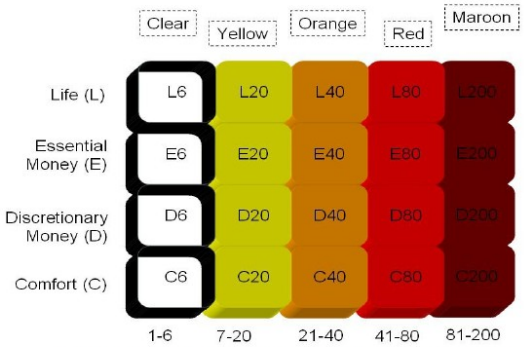

Figura 8: Código de colores en cristal

El código cromático se aplica a una forma tabular elaborada por Cockburn que se usa en muchos métodos ágiles para situar el rango de complejidad al cual se aplica una metodología. En la figura se muestra una evaluación de las pérdidas que puede ocasionar la falla de un sistema y el método requerido según este criterio. Los parámetros son:

- Comodidad (C),
- Dinero Discrecional (D),
- Dinero Esencial (E) y
- Vidas (L).

En otras palabras, la caída de un sistema que ocasione incomodidades indica que su nivel de criticalidad es C, mientras que si causa pérdidas de vidas su nivel es L. Los números del cuadro indican el número de personas afectadas a un proyecto, ±20%.

Los siete valores (propiedades) de Cristal

1. **Entrega frecuente:** a frecuencia dependerá del proyecto, pero puede ser diaria, semanal, mensual o lo que fuere. La entrega puede hacerse sin despliegue, si es que se consigue algún usuario que suministre feedback.
2. **Comunicación osmótica:** Todos juntos en el mismo cuarto.
3. **Mejora reflexiva:** Tomarse un pequeño tiempo para pensar bien que se está haciendo, cotejar notas, reflexionar, discutir.
4. **Seguridad personal:** Hablar cuando algo molesta.
5. **Foco.:**Saber lo que se está haciendo y tener la tranquilidad y el tiempo para hacerlo.
6. F**ácil acceso a usuarios expertos.**
7. **Ambiente técnico con prueba automatizada,** management de configuración e integración frecuente.

Estrategias comunes a otros métodos ágiles

1. **Exploración de 360°.** Verificar o tomar una muestra del valor de negocios del proyecto, los requerimientos, el modelo de dominio, la tecnología, el plan del proyecto y el proceso.
2. **Victoria temprana.** Es mejor buscar pequeños triunfos iniciales que aspirar a una gran victoria tardía.
3. **Esqueleto ambulante.** Es una transacción que debe ser simple pero completa. Podría ser una rutina de consulta y actualización en un sistema cliente-servidor, o la ejecución de una transacción en un sistema transaccional de negocios.
4. **Rearquitectura incremental.** Se ha demostrado que no es conveniente interrumpir el desarrollo para corregir la arquitectura, esta debe evolucionar en etapas, manteniendo el sistema en ejecución.

5. **Radiadores de información.** Es una lámina pegada en algún lugar que el equipo pueda observar mientras trabaja o camina.

Roles

1. **Patrocinador:** Produce la Declaración de Misión con Prioridades de Compromiso. Consigue los recursos y define la totalidad del proyecto.
2. **Usuario Experto:** Junto con el Experto en Negocios produce la Lista de Actores- Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación, etcétera.
3. **Diseñador Principal:** Produce la Descripción Arquitectónica. Se supone que debe ser al menos un profesional de Nivel 3. El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y programador más experto.
4. **Diseñador-Programador:** Produce, junto con el Diseñador Principal, los Borradores de Pantallas, el Modelo Común de Dominio, las Notas y Diagramas de Diseño, el Código Fuente, el Código de Migración, las Pruebas y el Sistema Empaquetado. Cockburn no distingue entre diseñadores y programadores.
5. **Experto en Negocios:** Junto con el Usuario Experto produce la Lista de Actores- Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe conocer las reglas y políticas del negocio.
6. **Coordinador:** Con la ayuda del equipo, produce el Mapa de Proyecto, el Plan de Entrega, el Estado del Proyecto, la Lista de Riesgos, el Plan y Estado de Iteración y la Agenda de Visualización.
7. **Verificador:** Produce el Reporte de Bugs. Puede ser un programador en tiempo parcial, o un equipo de varias personas.
8. **Escritor:** Produce el Manual de Usuario.

Desarrollo adaptativo de software (DAS)

El desarrollo adaptativo de software (DAS) fue propuesto por Jim Highsmith como una técnica para elaborar software y sistemas complejos. Los fundamentos filosóficos del DAS se centran en la colaboración humana y en la organización propia del equipo.
Highsmith argumenta que un enfoque de desarrollo adaptativo basado en la colaboración es “tanto una fuente de orden en nuestras complejas interacciones, como de disciplina e ingeniería”. Él define un “ciclo de vida” del DAS que incorpora tres fases: especulación, colaboración y aprendizaje.

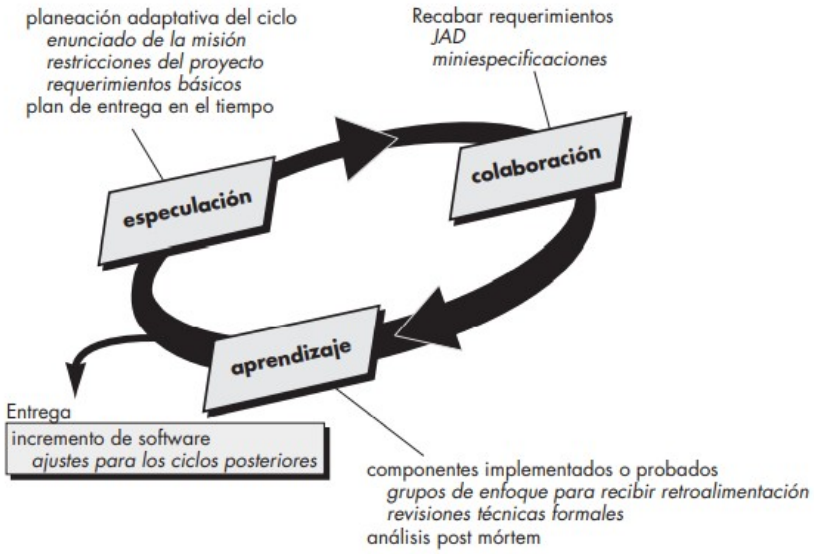

Figura 9: Modelo desarrollo adaptativo

Durante la especulación, se inicia el proyecto y se lleva a cabo la planeación adaptativa del ciclo. La especulación emplea la información de inicio del proyecto-enunciado de misión de los clientes, restricciones del proyecto (por ejemplo, fechas de entrega o descripciones de usuario) y requerimientos básicos para definir el conjunto de ciclos de entrega (incrementos de software) que se requerirán para el proyecto.

La colaboración no es fácil. Incluye la comunicación y el trabajo en equipo, pero también resalta el individualismo porque la creatividad individual desempeña un papel importante en el pensamiento colaborativo. Las personas que trabajan juntas deben confiar una en otra a fin de:

1. Criticarse sin enojo.
2. Ayudarse sin resentimiento.
3. Trabajar tan duro, o más, que como de costumbre.
4. Tener el conjunto de aptitudes para contribuir al trabajo.
5. Comunicar los problemas o preocupaciones de manera que conduzcan a la acción efectiva.

Por último el énfasis se traslada al “aprendizaje” de todo lo que hay en el avance hacia la terminación del ciclo, en este se revisa la calidad, y si no se tiene errores se entrega al cliente. Highsmith afirma que los desarrolladores de software sobreestiman con frecuencia su propia comprensión (de la tecnología, del proceso y del proyecto) y que el aprendizaje los ayudará a mejorar su nivel de entendimiento real. Los equipos DAS aprenden de tres maneras: grupos de enfoque revisiones técnicas y análisis post mórtem del proyecto.

Características

1. Interativo

2. Orientado a los componentes de Software
3. Tolerante a los cambios
4. Guiados por los riesgos

Ventajas

- Utiliza información disponible acerca de cambios para mejorar el comportamiento del software.
- Promulga colaboración, la iteración de personas.

Desventajas

- Los errores y cambios que no son detectados con anterioridad afectan la calidad del producto y su costo total.
- Ya que esta es una metodología ágil, no permite realizar procesos que son requeridos en las metodologías tradicionales.
- Sirve para aprender de los errores y volver a iniciar el ciclo del desarrollo.

Bibliografía

Calderón, M. A. (2017). DESARROLLO ADAPTATIVO DE SOFTWARE. issuu. https://issuu.com/cyber107/docs/metodologiaagilasd-100613135636-php

Calero, W., & Perfil, V. T. M. (s. f.). Modelo de Desarrollo Concurrente. Ingeniería de software. Recuperado 24 de noviembre de 2021, de http://ingenieraupoliana.blogspot.com/2010/10/modelo-de-desarrollo-concurrente.html

Guía de ingeniería del software, Instituto Nacional de tecnologías de la comunicación de España (INTECO) Pág. 30-32.

Mancuzo, G. (2021a, agosto 20). Metodología Kanban: Definición, Funcionamiento y Fases. ComparaSoftware Blog. Recuperado 27 de noviembre de 2021, de https://blog.comparasoftware.com/metodologia-kanban/

López, A. L. I. (2013, 23 mayo). Implementación de KANBAN en 4 fases. La logística no es TODO, pero esta en TODO. Recuperado 27 de noviembre de 2021, de https://logispyme.com/2013/05/24/implementacion-de-kanban-en-4-fases/

Mancuzo, G. (2021, 20 agosto). Scrum: Ventajas y Desventajas. ComparaSoftware Blog. Recuperado 27 de noviembre de 2021, de https://blog.comparasoftwareMancuzo, G. (2021a, agosto 20). Metodología Kanban: Definición, Funcionamiento y Fases. ComparaSoftware Blog. Recuperado 27 de noviembre de 2021, de https://blog.comparasoftware.com/metodologia-kanban/.com/scrum-ventajas-y-desventajas/

METODOS FORMALES. (2012, 8 febrero). innovacionpnfi2012. Recuperado 23 de noviembre de 2021, de https://innovacionpnfi2012.wordpress.com/metodos-formales-2/

Modelo Evolutivo. (s. f.). INGENIERIA DE SOFTWARE. Recuperado 23 de noviembre de 2021, de https://ingsoftware.weebly.com/modelo-evolutivo.html

Pressmas, R.S. (2005). Ingeniería del software, un enfoque practico. 7 edición.

Sommerville, I. (2005). Ingeniería del software. Pearson educación.

Stabile, A. G. (2020, 30 abril). Desarrollo ágil de software: Crystal Clear. Folder IT. Recuperado 28 de noviembre de 2021, de https://folderit.net/es/blog/desarrollo-agil-de-software-crystal-clear-es/