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.

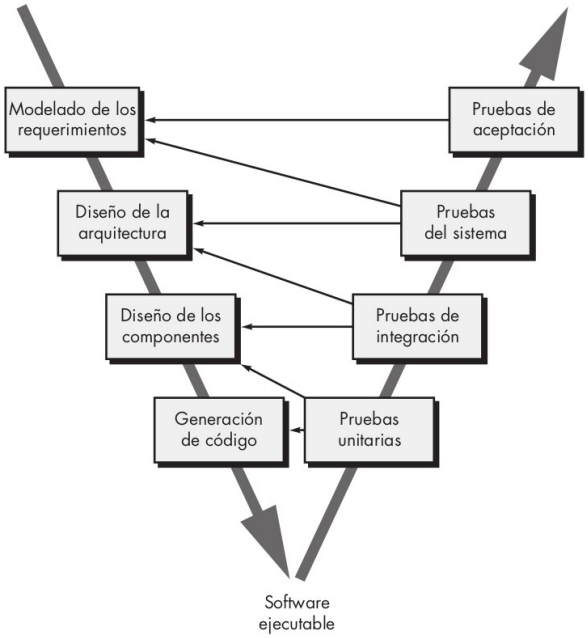

Figura 2: Modelo en V

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.

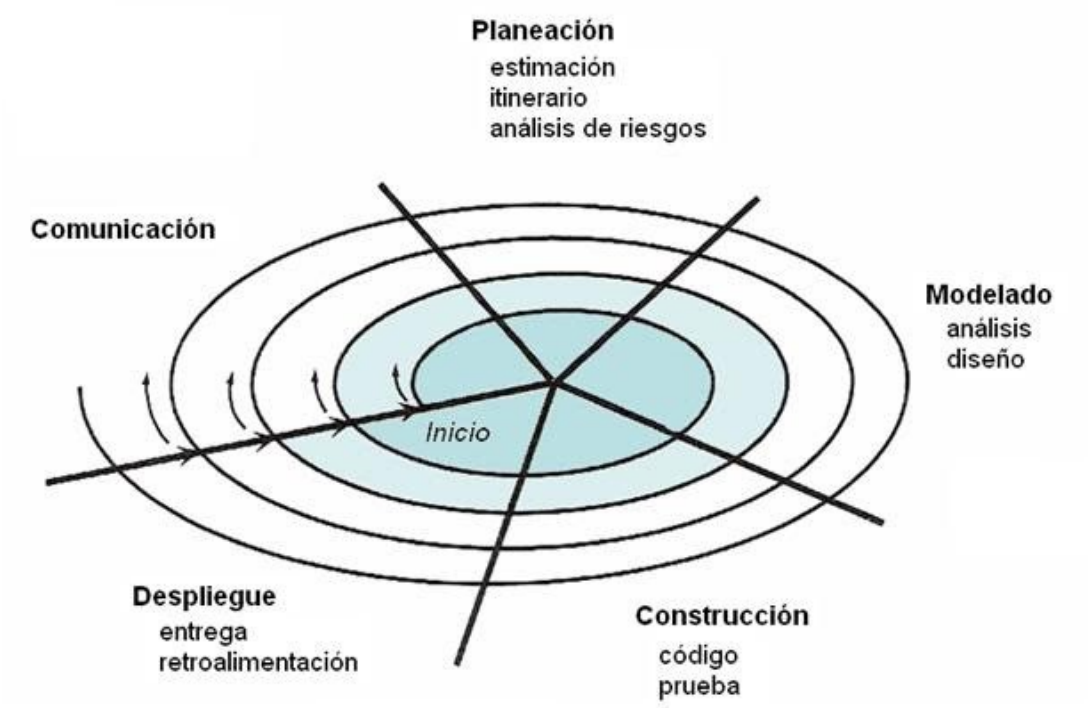

Figura 3: Modelo espiral

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.

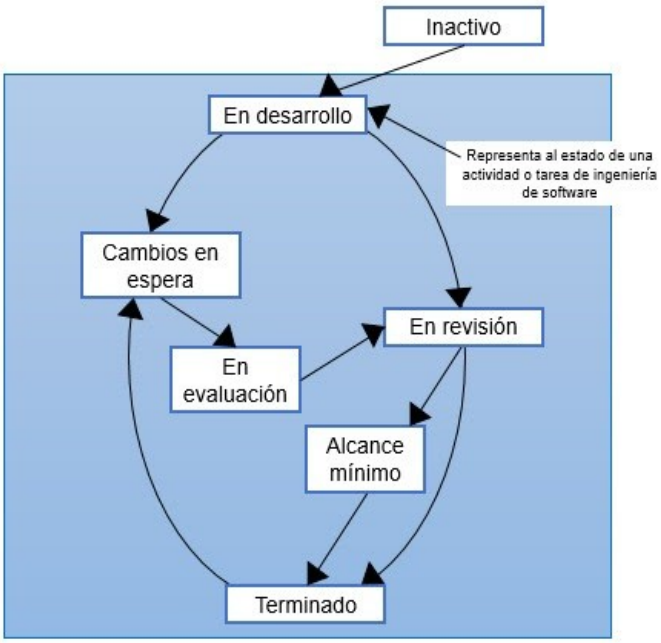

Figura 4: Modelo proceso concurrente

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.

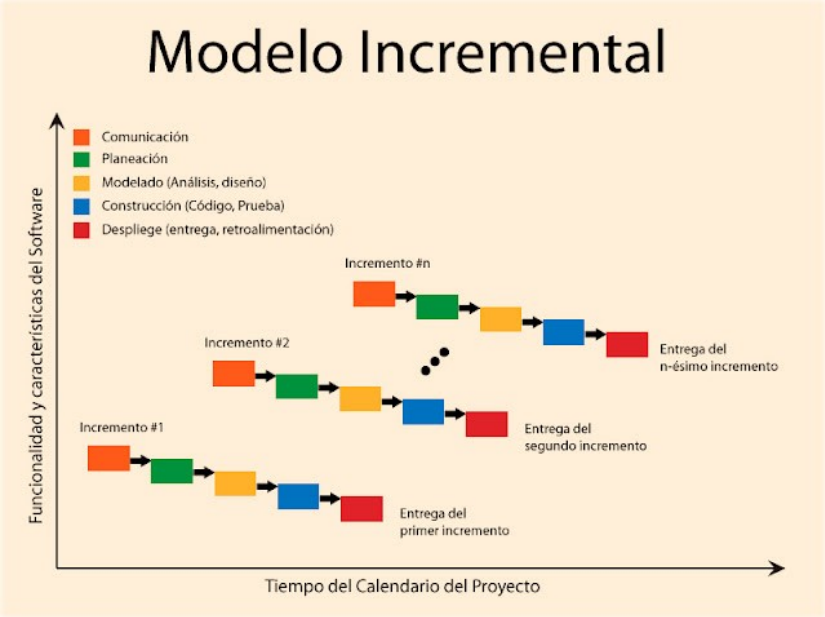

Figura 5: Modelo incremental

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.

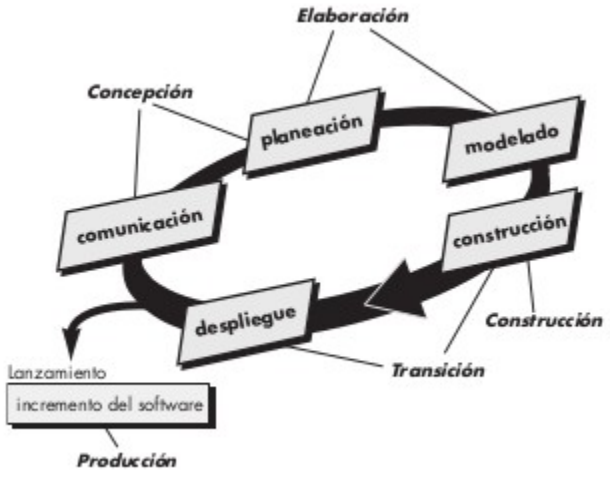

Figura 6: Modelo Proceso Unificado

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.

Bibliografía

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

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

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

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.