Técnico en
DESARROLLO DE SOFTWARE

Fundamentos de Construcción de Software

Fundamentos de Construcción de Software

Unidad I

1. Software

El Software es un conjunto de instrucciones que se ejecuta en una computadora para poder manipular y procesar datos que cumplen los requerimientos de una tarea específica. Por eso al software también se le denomina el componente lógico de un sistema informático, al contrario del hardware que es el componente físico. El software es una parte imprescindible de cualquier sistema informático y su importancia radica en permitir la interacción entre el hombre y la máquina.

El software es desarrollado utilizando distintos lenguajes de programación. Un lenguaje de programación es un conjunto de símbolos y reglas que representan instrucciones para las computadoras. Un lenguaje del nivel más bajo consiste en grupos de valores binarios, también llamado lenguaje máquina, que indican al procesador (CPU) cambiar el estado de la computadora. Este lenguaje es mucho más rápido para la computadora, ya que para el hardware es más fácil entender las instrucciones con unos y ceros. Para el ser humano es muy difícil escribir instrucciones en este tipo de lenguaje, por lo que se crearon los lenguajes de alto nivel que son similares a los lenguajes naturales. Los lenguajes de alto nivel son traducidos a lenguaje de bajo nivel por compiladores o

intérpretes antes de ejecutarse en la computadora. La mayoría del software está escrito en lenguajes de alto nivel, ya que son más fáciles y eficientes para los desarrolladores.

Según la definición formal del IEEE el Software es "El conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación"[1]. Tomando en cuenta esta definición, el software incluye desde las reglas del negocio hasta la documentación del sistema. La construcción o desarrollo de software se debe encargar de todos los procesos y aspectos relacionados al software.

Características

A diferencia de hardware que se fabrica, el software por ser un componente lógico del sistema informático tiene unas características muy particulares:

El software se desarrolla, no se fabrica

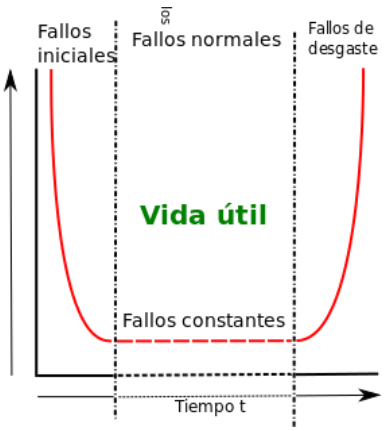
A pesar de que existen las similitudes entre el desarrollo de software y la fabricación de hardware, los dos procesos tienen diferencias fundamentales. En ambos casos existen fases de análisis, diseño y desarrollo o construcción, y la calidad del producto final depende de un buen diseño. Sin embargo, en la fase de producción del hardware pueden producirse problemas que afecten a la calidad y que no existen en el caso del software.

El software no se desgasta

[1] IEEE Software Engineering Standard: Glossary of Software Engineering Terminology. IEEE Computer Society Press, 1993

El software no se desgasta como el hardware, porque no sufre de los fenómenos físicos que afectan los componentes electrónicos. El software cuando se deteriora queda obsoleto debido a los cambios, correcciones y problemas de compatibilidad del entorno operativo.

Se puede representar la tasa de fallos de software con la forma de una "curva idealizada". Los defectos sin descubrir causan una alta tasa de fallos en las primeras etapas de vida de un programa. Sin embargo, los errores se corrigen y la curva se aplana, a esto se le denomina la vida útil del software. En la última etapa cuando la tasa de fallo sufre un crecimiento rápido indica que el software está deteriorado.



La mayoría del software se construye a medida

Las empresas buscan programas de software personalizados que tengan la capacidad de resolver eficientemente y simplificar problemas puntuales para que la sistematización ayude a reducir el gasto en las empresas. En algunos casos se pueden diseñar componentes del software reutilizables.

2. Evolución del Software

El software ha ido evolucionando a lo largo del tiempo, este se puede dividir en cinco eras. Algunos de los sucesos destacados han sido los siguientes:

Primer Era (1955-1965):

- Su principal desarrollo se definió con “Codificar y Corregir”
- Se desarrollaba sin un planteamiento previo.
- No existían fuentes de información de ningún tipo.
- Desarrollo a base de prueba y error.

Segunda Era (1965-1972)

- Se simplifico el código
- Aparición de tecnologías nuevas como el multiprocesamiento y las bases de datos
- Aparición del software como producto
- Inicio de la “Crisis del software”, donde muchos proyectos de software sobrepasaron el tiempo y el presupuesto estimado.
- Se definieron procedimientos para el desarrollo de software.

Tercera Era (1972-1985)

- Se introduce el concepto de “Sistemas Distribuidos”.
- Se crearon sistemas de información más complejos.
- Implementación de redes de área local y global.
- Se definieron metodologías para el diseño y análisis.

Cuarta Era (1985-2000)

- Impacto colectivo del Software
- Se implementan las redes de información, tecnologías orientadas a objetos
- Auge del internet y aplicaciones web
- Fortalecimiento de la seguridad informática

Quinta Era (2000-Presente)

- Aplicaciones móviles
- Se definieron metodologías ligeras para el desarrollo de software
- Inteligencia artificial
- Realidad virtual
- Realidad aumentada

3. Clasificación del Software

Existen 7 grandes categorías de software de computadora que plantean retos continuos a los desarrolladores de software:

Software de sistemas:

Conjunto de programas escritos para dar servicio a otros programas. Determinado software de sistemas (por ejemplo, compiladores, editores y herramientas para administrar archivos) procesa estructuras de información complejas pero deterministas.

Software de aplicación:

Programas aislados que resuelven una necesidad específica de negocios. Las aplicaciones en esta área procesan datos comerciales o técnicos en una forma que facilita las operaciones de negocios o la toma de decisiones administrativas o técnicas.

Software de ingeniería y ciencias:

Se ha caracterizado por algoritmos “devoradores de números. Las aplicaciones van de la astronomía a la vulcanología, del análisis de tensiones en automóviles a la dinámica orbital del transbordador espacial, y de la biología molecular a la manufactura automatizada.

Software incrustado:

Reside dentro de un producto o sistema y se usa para implementar y controlar características y funciones para el usuario final y para el sistema en sí. El software incrustado ejecuta funciones limitadas y particulares (por ejemplo, control del tablero de un horno de microondas) o provee una capacidad significativa de funcionamiento y control

Software de línea de productos:

Es diseñado para proporcionar una capacidad específica para uso de muchos consumidores diferentes. El software de línea de productos se centra en algún mercado limitado y particular o se dirige a mercados masivos de consumidores.

Aplicaciones web:

Esta categoría de software centrado en redes agrupa una amplia gama de aplicaciones. En su forma más sencilla, las webapps son poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas limitadas.

Software de inteligencia artificial:

Hace uso de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o con el análisis directo. Las aplicaciones en esta área incluyen robótica, sistemas expertos, reconocimiento de patrones (imagen y voz), redes neurales artificiales, demostración de teoremas y juegos.

4. Tendencias del mercado

Durante los primeros años de la era de las computadoras el software no se consideraba como un producto sino como un añadido al hardware. El desarrollo de software se realizaba sin ninguna planificación y no existían metodologías formales. Con el paso de tiempo eso trajo la repercusión a las organizaciones generando muchas fallas en el software desarrollado y elevando los costos de producción de software. Desde entonces el proceso de desarrollo de software ha sufrido grandes cambios adaptando distintos modelos de diseño, nuevas herramientas y tecnologías.

La tendencia de los métodos de desarrollo de software está orientada para lograr mayor productividad, interoperabilidad, usabilidad y mantenibilidad. La tendencia actual es el desarrollo ágil. El uso de una aproximación ágil permite a los equipos de desarrollo cambiar las prioridades del proyecto basadas en los comentarios del cliente o del mercado.

En cuanto al trabajo en equipo la tendencia son los equipos de trabajo distribuidos, ahora con la mejora de la conexión a internet es más fácil formar equipos con desarrolladores que se encuentren en cualquier parte del mundo. Incluso se puede disminuir el costo de un proyecto trabajando con desarrolladores remotos.

El Software en Guatemala

Actualmente Guatemala a pesar de muchos desafíos cuenta con un buen número de empresas relacionadas con el desarrollo de software que han logrado éxito y han ganado reconocimiento. A nivel regional y global existe una alta demanda de los servicios de desarrollo de software de aplicaciones, de diseño y desarrollo web, publicidad en Internet entre otros. A muchas empresas del extranjero les resulta más conveniente subcontratar empresas de Centroamérica por los bajos costos de operaciones. Guatemala tiene oportunidad de ser un país más competitivo en el área de desarrollo de software y aumentar la exportación de sus productos al extranjero. El país cuenta con la Comisión de Software de Exportación SOFEX que está conformada por empresas que se dedican al desarrollo de software y ofrecen productos y servicios de alta calidad al mercado local e internacional.

5. Proceso del Software

Mitos de la construcción de software

A principios de los años 70 debido a las dificultades presentadas en el desarrollo de software frente al rápido crecimiento de la demanda por software, por la complejidad de los problemas a ser resueltos y de la inexistencia de técnicas establecidas para el desarrollo de sistemas adecuados surgió el término “crisis del software”. Básicamente este término se refiere a la dificultad de escribir los programas libres de defectos.

Los problemas presentados en el desarrollo de software se centran sobre los aspectos:

1. La planificación y estimación de costes son frecuentemente imprecisas.

2. La productividad de la comunidad de software no se corresponde con la demanda de sus servicios.
3. La calidad de software no llega a ser aceptable.

Estos problemas se han producido por el propio carácter del software y por los errores de las personas encargadas del desarrollo del mismo.

Hay que recordar que el software es un elemento lógico y el éxito se mide por la calidad del software como un todo no por la cantidad de software fabricado. El mantenimiento de software incluye normalmente la corrección o modificación del diseño.

Muchas de las causas de crisis del software se pueden encontrar en una mitología que surge durante los primeros años del desarrollo de software.

Mitos de gestión

Los gestores normalmente trabajan bajo presión de cumplir con el presupuesto, hacer que no se retrase el proyecto y mejorar la calidad. A veces tienen ideas erróneas sobre su equipo de trabajo, algunos de esos mitos son los siguientes:

Mito: Existe ya un libro lleno de estándares y procedimientos para construir software. ¿Esto le proporciona al programador todo lo que necesita?

Realidad: La respuesta es no, los procedimientos dependen del tipo de proyecto que se esté trabajando.

Mito: Los programadores deben tener las herramientas de desarrollo más avanzadas

Realidad: Se necesita mucho más que la computadora último modelo para el desarrollo

de software de calidad.

Mito: Se puede añadir más programadores y adelantar el tiempo perdido.

Realidad: El desarrollo de software no es un proceso mecánico de fabricación. Añadir gente a un proceso de desarrollo de software puede implicar retrasar aún más el proyecto porque se debe capacitar de los nuevos miembros.

Mitos del cliente

En muchos casos el cliente cree en los mitos sobre el software. Esto conlleva que el cliente puede crear falsas expectativas sobre el proyecto.

Mito: Una declaración general de los objetivos es suficiente para escribir los programas.

Realidad: Una mala definición desde el principio es la causa principal de fallas de software. Debe existir una buena comunicación entre el cliente y los analistas para evitar errores en el diseño desde el principio.

Mito: los requisitos del proyecto cambian continuamente, pero los cambios se acomodan fácilmente debido a que software es flexible.

Realidad: El impacto varía dependiendo del momento en el que se introduce el cambio. Las modificaciones solicitadas al principio del proyecto pueden acomodarse fácilmente. El cliente puede revisar los requisitos y recomendar los cambios con relativamente poco impacto en el coste. Conforme pasa el tiempo el impacto en el coste crece con rapidez.

Mitos de los desarrolladores

Los mitos han ido fomentando durante los años en la cultura informática. En la primera década del desarrollo de software la programación se veía como un arte. Y las viejas formas y actitudes es difícil cambiar.

Mito: una vez que se desarrolló el programa y está funcionando, la labor del desarrollador ha terminado.

Realidad: Se ha comprobado que el trabajo de programación se realiza aún después de que se le haya entregado al cliente.

Mito: Hasta que el programa no se está ejecutando, no hay forma de comprobar su calidad.

Realidad: Desde las etapas iniciales del proyecto se puede aplicar la revisión técnica formal para encontrar algunas deficiencias en el sistema.

Mito: Al finalizar el proyecto lo único que se entrega es el programa funcionando.

Realidad: Un programa funcional es solo una parte de la configuración del software que incluye mucho más elementos tales como un plan, las especificaciones de requisitos, las estructuras de datos, listados y especificaciones de la prueba.

Estos mitos y las actitudes viejas fomentan la mala gestión y malas prácticas en el proceso de construcción de software. Para poder solucionar los problemas en el desarrollo del software el primer paso es reconocer las realidades del mismo.

6. Ingeniería de Software

Para solucionar los problemas en el desarrollo de software no existe una única solución. Sin embargo con una combinación de métodos para todas las etapas del desarrollo, herramientas automatizadas, técnicas para garantizar la calidad del software y una filosofía predominante para la coordinación, control y gestión se puede conseguir una disciplina para el desarrollo de software: Ingeniería de Software.

La Ingeniería de software surge de Ingeniería de sistemas y de hardware y contiene tres elementos clave: métodos, herramientas y procedimientos.

Los **métodos** indican cómo construir el software. Describen las tareas de planificación y estimación de proyectos, análisis de los requerimientos, diseño de la estructura de datos, arquitectura de programas y algoritmos, codificación, prueba y mantenimiento.

Las **herramientas** proporcionan un soporte automatizado para la implementación de los métodos. La ingeniería del software asistida por computadora, en inglés Computer Aided Software Engineering CASE, combina software, hardware y bases de datos para crear un entorno de ingeniería del software.

Los **procedimientos** indican la secuencia en que se aplican los métodos usando las herramientas para facilitar un desarrollo racional y oportuno del software.

7. Procesos de la construcción de software

El desarrollo de software puede simplificarse en gran medida si se definen y utilizan procesos que luego pueden estandarizarse. Es importante definir los siguientes conceptos:

- **Proceso:** Un conjunto de actividades que deben realizarse para crear un producto.
- **Actividad:** Es cualquier acción claramente definida hacia un objetivo a grandes razgos del proyecto, sin importar el área objetivo del proyecto. Es decir, no necesariamente debe de limitarse al desarrollo del proyecto sino puede tomar cualquier entorno de la aplicación. Por ejemplo una reunión con el equipo de trabajo.
- **Acción:** La realización de tareas que producen un avance en el proyecto. Por ejemplo un modelo de la arquitectura del sistema.
- **Tarea:** Se enfoca en un objetivo pequeño, pero bien definida. Un ejemplo puede ser el desarrollo y ejecución de una prueba unitaria.

En la realidad, la definición de un proceso no siempre es inquebrantable. Se pueden adaptar los procesos de desarrollo de software a cada proyecto para de esta manera elegir las acciones y tareas apropiadas. El proceso del software siempre busca entregar el producto en el tiempo estipulado y de buena calidad. En resumen un proceso define las acciones de cada individuo, el momento en que deben de realizarse y de qué manera para poder alcanzar el objetivo, un producto terminado.

Es importante definir la estructura del proceso ya que es el fundamento para identificar las actividades estructurales, que son los bloques con los que se desarrolla el software pero también es de suma importancia definir las actividades administrativas, de seguimiento y control del proyecto. Las actividades administrativas pueden cambiar el flujo del proyecto de manera significativa, por ejemplo una diferencia en legislaciones de región a región. Esencialmente una estructura de proceso de ingeniería de software

consta de cinco actividades: la comunicación, planeación, modelado, construcción y despliegue. Usualmente estas actividades se aplican en ese orden y se repiten con cada proyecto nuevo. Este enfoque puede utilizarse también dentro del proyecto para el desarrollo de cada módulo de software y que de esta manera realiza incrementos en los entregables del proyecto. A continuación se detallan estos componentes de la estructura del proceso.

Comunicación: Este componente es indispensable y es la primera acción que debe realizarse. La primera acción es comunicarse con el cliente para obtener requerimientos y objetivos del proyecto.

Planeación: Es importante tanto para el desarrollo como para la gestión del proyecto. Sirve de guía para conocer los pasos, tareas y acciones que deben de realizarse y en qué orden realizarlos. Este componente también se conoce como el "plan de proyecto de software". Además de los pasos, tareas y acciones que se definen en este documento, también se incluyen los análisis de riesgos, los recursos requeridos, el resultado del trabajo y una calendarización de las actividades.

Modelado: Son representaciones del producto terminado que se utilizan para ejemplificar el producto final o algunas funcionalidades y características del producto al cliente y son usadas por los desarrolladores como guía. Esto evita malos entendidos y disminuye el reproceso de actividades y malgasto de tiempo innecesario.

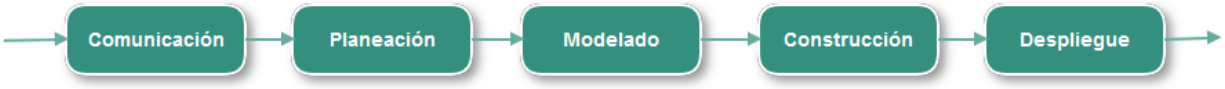
Construcción: El proceso de generar el código y pruebas de control de calidad para obtener el producto entregable final.

Despliegue: Depende de cómo se maneje el proyecto, puede ser un entregable parcial para un incremento en la funcionalidad o el producto terminado. Este paso es donde se hace una entrega formal al cliente para que lo evalúe, otorgue su retroalimentación, realizar mínimos ajustes según sea necesario y el software sea puesto en el ambiente de producción.

8. Flujos de proceso de software

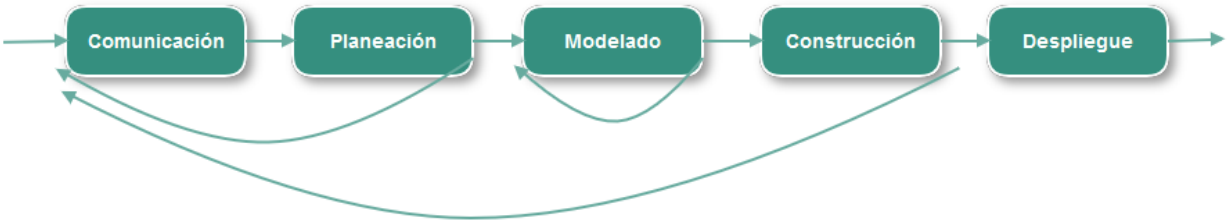
El flujo del proceso de software define el orden en que los cinco componentes de la estructura del proceso deben llevarse a cabo. Existen cuatro distintos tipos de flujo de proceso. Es importante notar que en cada cambio de etapa, pueden aplicarse diversas actividades como seguimiento y control de proyecto, administración de riesgos, aseguramiento de calidad, etc. En el flujo de proceso de software se definen los productos a obtener conforme el trabajo que se realizará.

- **Flujo de proceso lineal:** Lleva a cabo las actividades de manera secuencial hasta llegar al producto terminado. El orden en que se realizan las actividades es el siguiente:

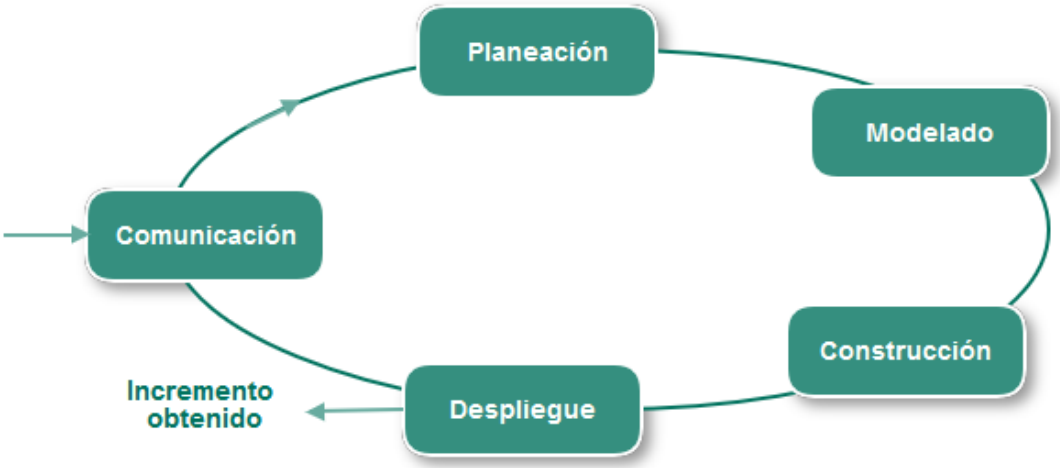


- **Flujo de proceso iterativo:** Este tipo de flujo se diferencia en que se repiten las diferentes actividades para realizar correcciones bajo la marcha si fuera necesario. En la vida real es cuando se involucran a los distintos miembros del equipo de desarrollo o incluso a los clientes en diferentes etapas de desarrollo, pero no se realiza ningún entregable hasta la fase de despliegue. La acción de

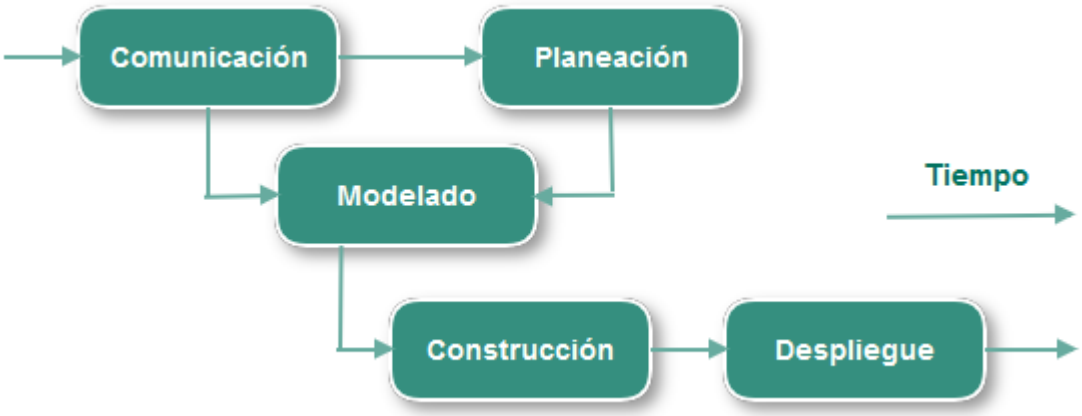
comunicación puede llevar a una nueva etapa de planificación, cambios en el modelado y los cambios y avances quedan plasmados en la etapa de construcción y cuando esto sucede, puede causar que la actividad de comunicación se realice nuevamente hasta llegar al punto donde el producto sea terminado y desplegado a producción.



- **Flujo de proceso evolutivo:** En este flujo, las actividades se repiten varias veces, en orden y cada vez entregando versiones con mayor funcionalidad del programa hasta que sea finalizado el producto.



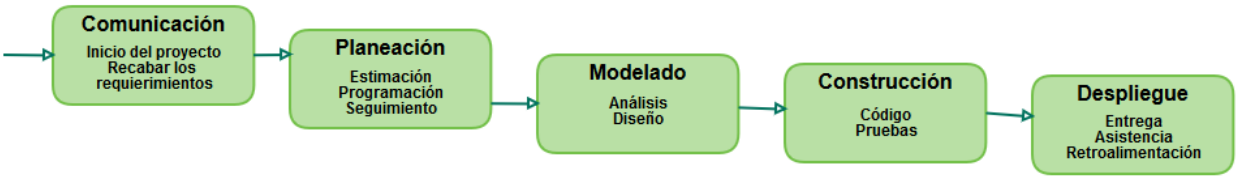
- **Flujo de proceso paralelo:** Este flujo se caracteriza por ejecutar actividades en paralelo, de manera simultánea. Por ejemplo, la planeación de un componente del software puede ser ejecutada de manera simultánea que el modelado y construcción de componentes diferentes del mismo software.



9. Modelos de proceso o ciclo de vida

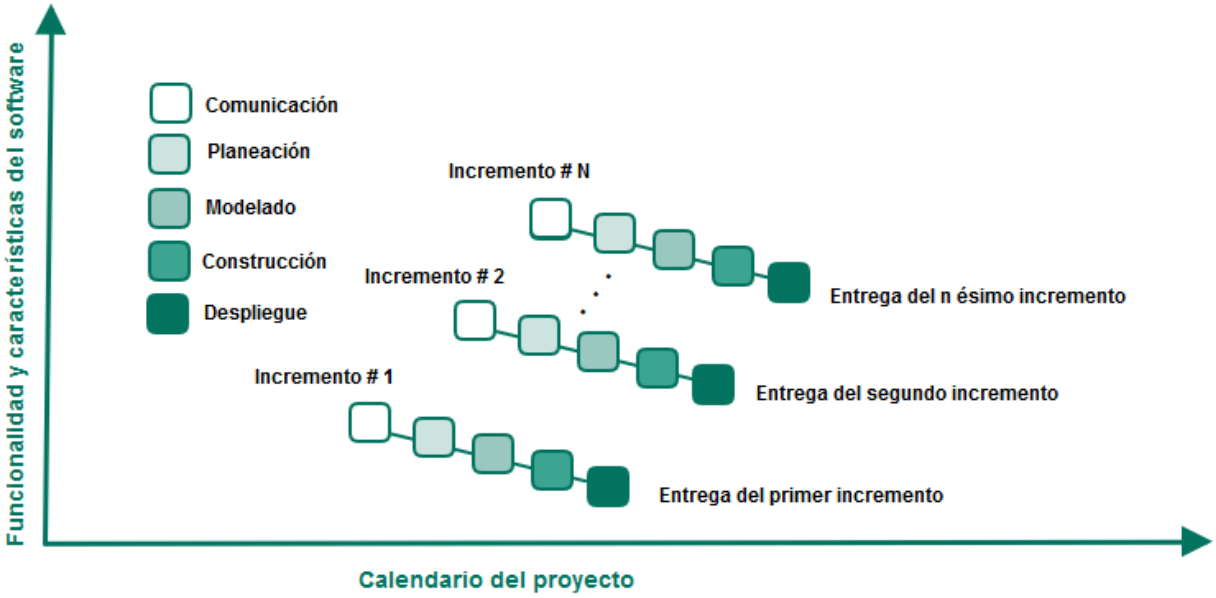
Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software, determina el orden de las etapas involucradas y los criterios de transición asociados entre estas etapas. A continuación se describen algunos de los enfoques que han probado ser exitosos.

- **Modelo de cascada:** Este enfoque también se conoce como el ciclo de vida clásico de software, ya que su enfoque es llevar a cabo las tareas como una secuencia. Una tarea no puede iniciar si la anterior no ha terminado. Inicia con la tarea de comunicación y finaliza con el despliegue. Este enfoque se utiliza cuando los requerimientos no cambian y están bien definidos.



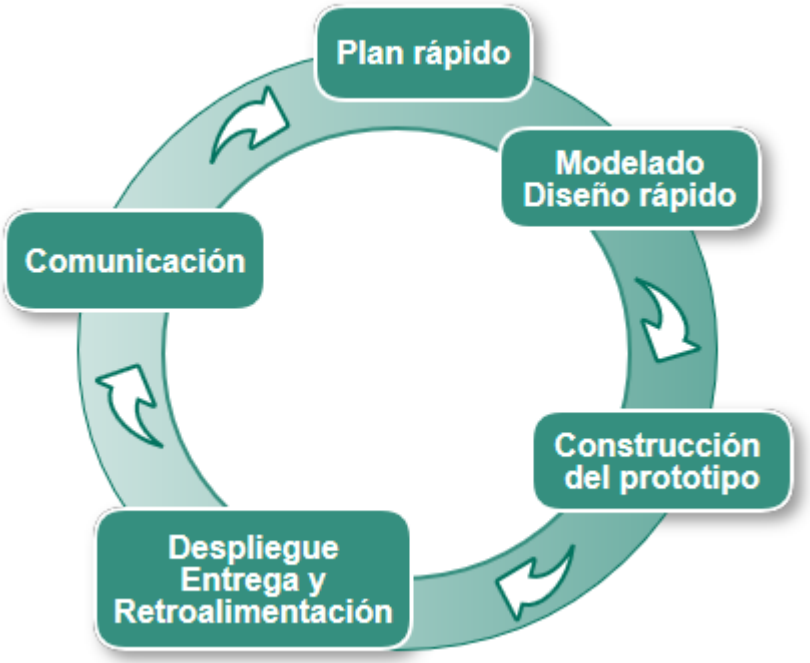
- **Modelo de proceso incremental:** Este modelo es utilizado cuando está definido una parte del sistema, pero existen factores que no permiten que el desarrollo de

software sea de manera lineal. Es una mezcla de proceso lineal y en paralelo, es decir: Se puede iniciar el desarrollo de manera lineal hasta tener un software básico, luego se realiza la programación de los siguientes módulos o incrementos para agregar funcionalidades al proyecto base. Un ejemplo simple sería el de una calculadora donde la primera etapa sea el desarrollo de una calculadora que sume dos números. Luego se desarrollan por incrementos los módulos de operaciones básicas (resta, multiplicación, división, etc.), módulo de almacenamiento de memoria, funciones de exponentes, etc. Cada incremento es planeado para adecuarse a las necesidades del cliente y proporcionar características y funcionalidad incrementada.



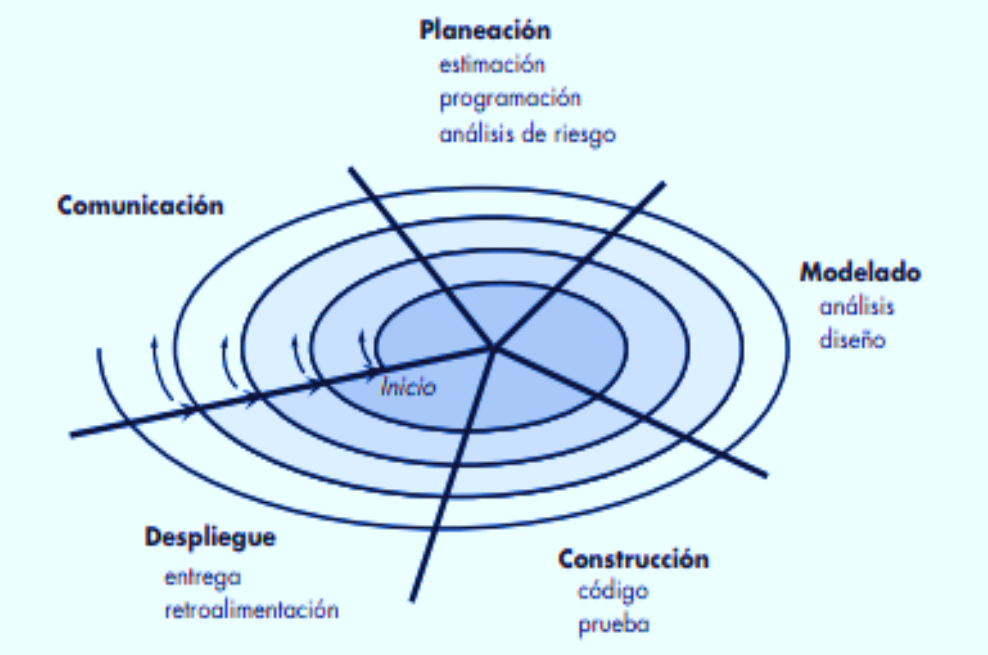
- **Modelo de prototipos:** Este modelo suele ser utilizado donde el tiempo de lanzamiento es crítico, debe de realizarse en el menor tiempo posible, incluso cuando no se hayan creado todos los componentes del sistema. Se aíslan diferentes componentes y se trabajan en el orden que sea necesario y al momento de despliegue son nuevas “versiones” del producto. Por ejemplo, un cliente

requiere un sistema de rifas que permita el registro de usuarios y al final de cierto período se realice un sorteo. Es sumamente importante que los usuarios se registren lo antes posible para tener una base de clientes grande, por lo cual se puede hacer el prototipo que solo permite el registro, luego en la siguiente versión, el software permite la generación de un ganador aleatorio y así sucesivamente. Este modelo se caracteriza por realizar las acciones rápidamente y adaptarse sobre la marcha.



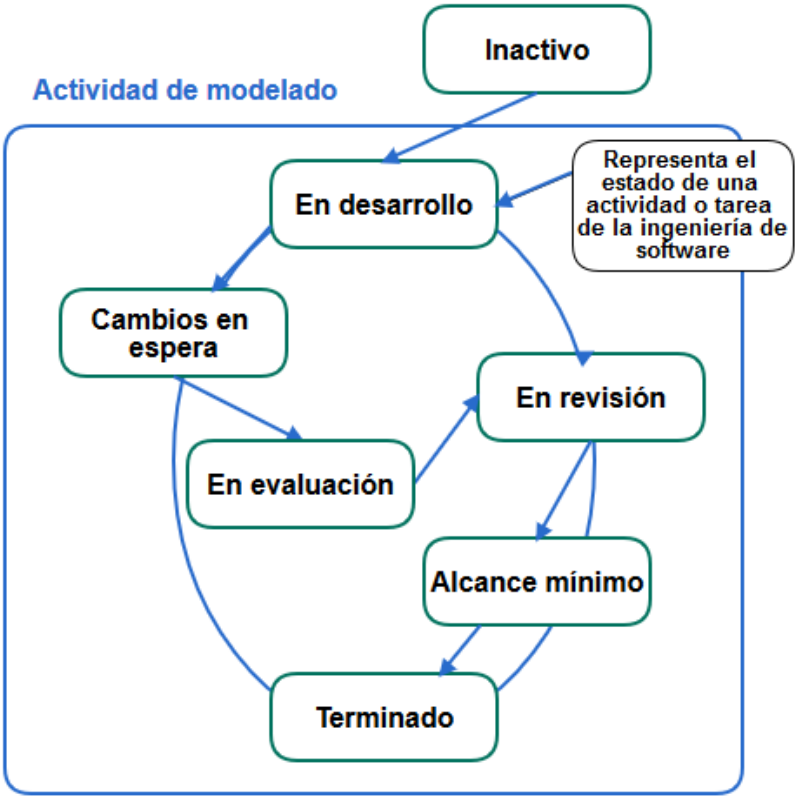
Este modelo suele emplearse cuando se presentan los prototipos a los clientes, y estos piensan que el sistema presentado ya es una versión funcional del software, por lo cual los desarrolladores construyen sobre el sistema y muchas veces no contienen buenas prácticas de programación y utilizan cualquier tiempo de atajos para lograr que el software sea funcional en el menor tiempo posible.

- **Modelo en espiral:** Es una combinación del modelo de cascada y el de prototipos. La diferencia es que cada vez que se realice un ciclo del modelo de cascada, el prototipo tiene mayor funcionalidad. Otro factor clave es que se subdividen las tareas de todo el sistema y se escogen puntos clave para los cuales se aplican las actividades de comunicación, planeación, modelado, construcción y despliegue.



- **Modelo concurrente:** Este modelo usualmente es utilizado en proyectos en los cuales varios equipos de trabajo se ven involucrados. Por ejemplo, un equipo de trabajo realiza un API donde se obtienen detalles de productos de una tienda y otro equipo realiza la interfaz gráfica que utilizan los usuarios. Debido a que son tareas separadas, esto permite que los diversos equipos utilicen los modelos descritos anteriormente, sin importar que sean diferentes. Al final, debe agregarse

tareas de integración, pero son una muy buena forma de dividir el trabajo en equipos especializados, para obtener un producto de mayor calidad en menor tiempo. Esto también puede ser malo si un equipo de trabajo se atrasa y fuerza que todo el proyecto tenga retrasos y esto incrementa los costos de desarrollo.



- **Modelo basado en componentes:** Es muy similar al modelo de espiral, pero en este modelo, el objetivo de cada iteración, es la generación de distintos componentes de software genéricos, reutilizables que pueden interconectarse o funcionar independientemente. Para lograr esto es de suma importancia evaluar el tipo de aplicación y si es candidata para generar distintos componentes y cómo deben integrarse unos con otros. Por ejemplo, en el sistema de un establecimiento

educativo, el componente de notas de estudiante puede desarrollarse de manera independiente que el componente de facturación.

10. Metodologías de desarrollo de software

Desarrollo rápido de aplicaciones (RAD)

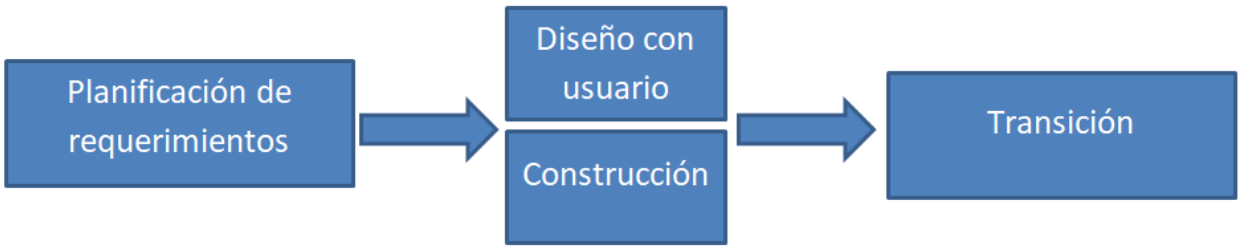
La metodología de desarrollo rápido de aplicaciones (RAD) fue formalizada por James Martin en 1991 como una alternativa a la metodología de cascada. Esta metodología cambia el enfoque de la planificación costosa a la creación de prototipos por medio de herramientas de ingeniería de software asistidas por computadora. Por ejemplo, para obtener una página web de gestión de contenido simple, se pueden utilizar herramientas donde el desarrollador construye el sitio web arrastrando y soltando componentes. Estos componentes que han sido desarrollados previamente se combinan y crean productos de software de alta calidad, reduciendo el tiempo y el costo de desarrollo.

RAD funciona muy bien para pequeñas y medianas empresas, donde los usuarios comerciales son dueños del presupuesto y están motivados para obtener resultados. Un ejemplo clásico son aplicaciones de línea de negocio, un término general que describe a las aplicaciones desarrolladas para automatizar y ejecutar ciertas partes del negocio de manera más eficiente. Del mismo modo, RAD es perfectamente aplicable para la creación de sitios web. Por lo general, estos son proyectos pequeños con un conjunto limitado de partes interesadas, pero involucrarlos temprano es crucial ya que la mayoría tiene una opinión al respecto.

RAD es un concepto antiguo, pero hoy está experimentando un renacimiento siguiendo las tendencias de la transformación digital y el impulso hacia un tiempo de comercialización más rápido.

Pasos o fases de la metodología RAD

1. **Planificación de requerimientos:** Durante este paso, las partes interesadas se sientan juntas para definir y finalizar los requisitos del proyecto, como los objetivos, las expectativas, los plazos y el presupuesto del proyecto.
2. **Diseño con el usuario:** Los diseñadores y desarrolladores trabajarán en estrecha colaboración con los clientes para crear y mejorar los prototipos en funcionamiento hasta que el producto final esté listo.
3. **Construcción:** Tan pronto como termine de definir el alcance del proyecto, se puede comenzar el desarrollo. Este paso requiere que pruebe su producto de software y se asegure de que todas sus partes móviles funcionen juntas según las expectativas del cliente.
4. **Transición:** Este es el paso final antes de que se lance el producto terminado. Implica la conversión de datos y la capacitación del usuario.

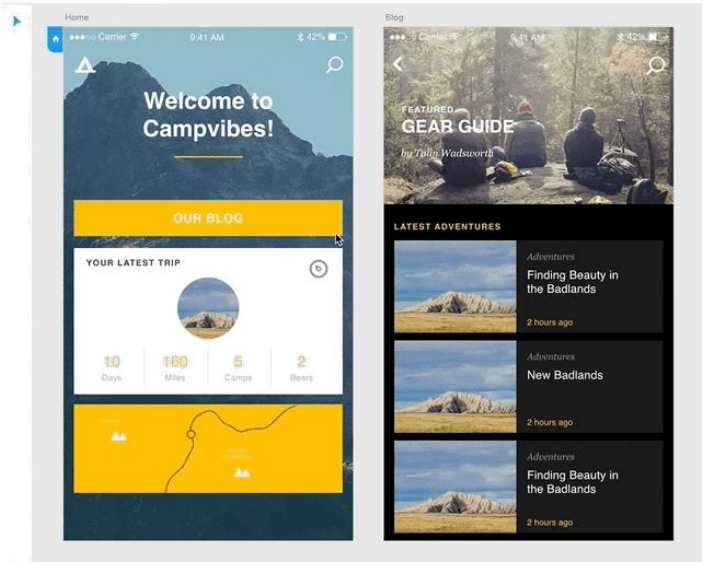


Herramientas

Elegir las herramientas correctas es fundamental para garantizar la creación rápida de prototipos y, por lo tanto, la implementación exitosa de la metodología RAD en un proyecto. Afortunadamente, existe una amplia variedad de herramientas y plataformas dirigidas a diferentes tipos de aplicaciones, etapas de un proyecto y habilidades de equipo.

Diseño y prototipado

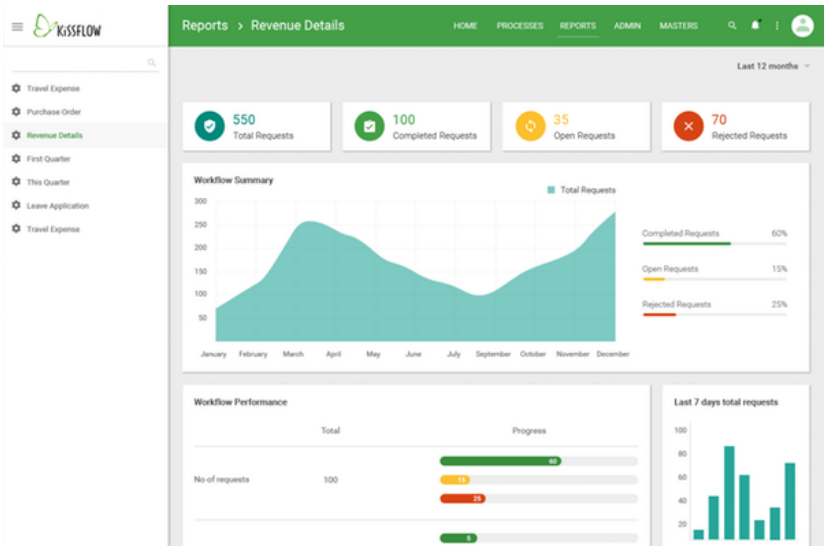
Las herramientas como Figma, InVision y Adobe XD permiten a los diseñadores gráficos y especialistas en usabilidad construir rápidamente prototipos y compartir los diseños completos con los usuarios finales para recopilar sus comentarios. Una vez que se aprueba una iteración del prototipo, el proyecto puede exportarse a formatos reutilizables por los desarrolladores, pasando así a la fase de construcción. Estas herramientas son las más utilizadas para crear sitios web, pero también se pueden usar para crear prototipos de la experiencia del usuario de aplicaciones o portales de usuarios más complejos.



Construcción

Plataformas de código bajo o sin código

La idea clave de estas plataformas es permitir a los usuarios empresariales sin habilidades de desarrollo (conocidos como usuarios avanzados o desarrolladores ciudadanos) entregar aplicaciones de trabajo muy rápidamente. Por supuesto, esta simplicidad implica falta de flexibilidad y múltiples limitaciones. Como resultado, el nicho de tales plataformas es la creación de prototipos o sistemas muy básicos. Algunos ejemplos de estas plataformas son Wix, Mendix, Kissflow.



Plataformas enfocadas en el desarrollador

Estas plataformas proporcionan herramientas de generación de código, por lo que los desarrolladores se mantienen alejados de escribir código repetitivo y les brinda funcionalidades que les facilita su trabajo. Algunos ejemplos son: PHPRad y Ruby on Rails.

Proceso Racional Unificado

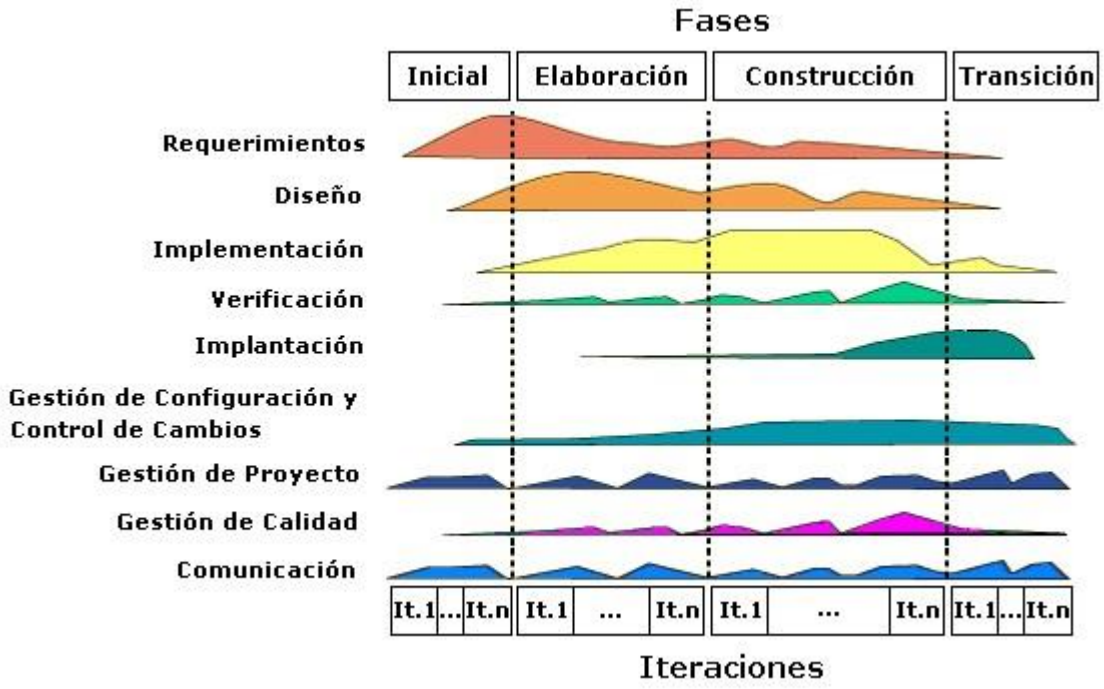
La metodología de *Proceso Racional Unificado* fue creada por Rational una división de IBM, proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es garantizar la producción de software de alta calidad que satisfaga las necesidades de sus usuarios finales, dentro de un calendario y presupuesto predecible. Es un proceso considerado pesado y preferiblemente aplicable a grandes equipos de desarrollo y grandes proyectos.

Esta metodología proporciona a cada miembro del equipo las pautas, plantillas y herramientas necesarias para que todo el equipo aproveche al máximo las siguientes mejores prácticas:

1. Desarrollar software iterativamente
2. Gestionar requisitos
3. Usar arquitecturas basadas en componentes
4. Utilizar software de modelado visual
5. Verificar la calidad del software
6. Control de cambios en el software

El proceso se puede describir en dos ejes:

- El eje horizontal representa el tiempo y muestra el aspecto dinámico del proceso a medida que se ejecuta, y se expresa en términos de ciclos, fases e iteraciones.
- El eje vertical representa el aspecto estático del proceso: cómo se describe en términos de actividades, artefactos, trabajadores y flujos de trabajo.



Como se puede apreciar en la gráfica el tiempo que se invierte en cada actividad cambia a lo largo de la vida del proyecto. Por ejemplo, en las primeras iteraciones se dedica más tiempo a los requisitos, y en las iteraciones posteriores se invierte más tiempo en la implementación. Las actividades cómo gestión de configuración, gestión de proyecto y gestión de calidad se realizan durante todas las iteraciones del proyecto.

Las fases proporcionan hitos comerciales bien definidos que aseguran que las iteraciones progresen y converjan en una solución, en lugar de simplemente iterar indefinidamente. Todas las fases generan artefactos. Estos se utilizarán en la siguiente fase y documentarán el proyecto y permitirán un mejor seguimiento.

Fases

Fase inicial

La fase inicial contiene los flujos de trabajo necesarios para el acuerdo de las partes interesadas con los objetivos, la arquitectura y la planificación del proyecto. Si estos actores tienen buenos conocimientos, no será necesario hacer muchos análisis. En esta etapa, los requisitos esenciales del sistema se transforman en casos de uso. Este paso generalmente es corto y se utiliza para definir si es factible continuar con el proyecto y definir los riesgos y los costos. Se puede hacer un prototipo para que el cliente lo apruebe.

Fase de elaboración

El propósito de la fase de elaboración es analizar el dominio del problema, desarrollar el plan del proyecto y eliminar los elementos de mayor riesgo del proyecto. Para lograr estos objetivos, debe tener una vista amplia del problema a resolver. Las decisiones deben tomarse con una comprensión de todo el sistema: su alcance, funcionalidad principal y los requisitos de rendimiento.

Fase de construcción

En la fase de construcción, comienza el desarrollo del software, códigos de producción, pruebas alfa. Las pruebas beta se llevan a cabo al comienzo de la fase de transición.

Fase de transición

En esta fase los productos (lanzamientos, versiones) se entregarán al cliente. En esta etapa también tiene lugar la capacitación de los usuarios y el monitoreo.

Metodología ágil

La metodología ágil es un enfoque para el desarrollo de software donde la interacción del equipo, la colaboración con el cliente y la respuesta al cambio son los temas clave. Además de eso, la metodología ágil proporciona un marco en el que se producen mejoras continuas en las diferentes etapas del ciclo de vida del desarrollo de software.

A principios de la década de 1990, cuando las computadoras comenzaron a ingresar en las organizaciones, el desarrollo de software se enfrentó a una crisis llamada "crisis de desarrollo de aplicaciones" o " retraso en la entrega de aplicaciones". Lo que, a su vez, significaba que el tiempo entre definir una necesidad y obtener una aplicación real era en promedio de aproximadamente tres años. Esto implicaba que, al entregar los resultados los requisitos del cliente cambiaron o el valor del producto disminuyó por lo que todo el proceso resultaba demasiado costoso.

En 2001 un pequeño grupo de gurús del software, cansados del enfoque tradicional, se reunieron y escribieron un manifiesto que se convirtió en un principio rector para el desarrollo de software ágil.

La declaración central del Manifiesto Ágil dice: "Estamos descubriendo mejores formas de desarrollar software haciéndolo y ayudando a otros a hacerlo". A través de este trabajo, hemos llegado a valorar:

- Individuos e interacciones sobre procesos y herramientas.
- Software de trabajo sobre documentación completa
- Colaboración del cliente sobre negociación de contrato
- Responder al cambio sobre seguir un plan

Manifiesto ágil

Los siguientes son los 12 principios principales de la metodología ágil:

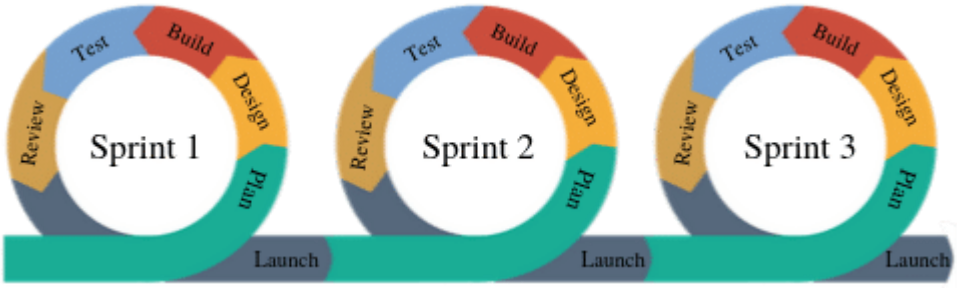
1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Marcos de trabajo

Para poder trabajar de forma estructurada surgieron diferentes marcos de trabajo bajo esta metodología, algunos se centran en la implementación (por ejemplo, Xtreme Programming), mientras que otros se centran en el flujo de trabajo (por ejemplo, Scrum y Kanban), mientras que otros buscan cubrir todo el ciclo de vida (Proceso Unificado Ágil).

Todos los marcos de trabajo ágiles funcionan de forma similar con ciclos de trabajo llamados Sprint que se ejecutan de forma iterativa.



Pero desde el punto de vista de la implementación, cada metodología tiene su propio conjunto de prácticas, terminologías, etc. Los marcos de trabajo más populares son los siguientes:

Scrum

Scrum es una de las metodologías ágiles más utilizadas. Ha ganado popularidad dentro de la comunidad de desarrollo de software porque es simple y tiene una tasa de productividad comprobada.

Scrum se centra en el equipo. Es un método que se concentra específicamente en cómo administrar tareas dentro de un entorno de desarrollo basado en equipo. Scrum cree en habilitar y capacitar al equipo de desarrollo y sugiere trabajar en equipos pequeños (por ejemplo, de 5 a 9 miembros).

El principio fundamental de Scrum es que, al dividir el tiempo y los proyectos, puede mejorar la eficacia y la productividad de una organización.

Scrum hace que el proceso de desarrollo sea menos complicado al hacer que la información sea transparente. Lo que, a su vez, ayuda al equipo a superar las desventajas del modelo en cascada como:

- Subestimación y sobreestimación del tiempo
- Falta de informe de progreso
- Incapacidad para adaptarse a los cambios, etc.

Roles en Scrum

Consiste en tres posiciones:

- Scrum Master: Es un facilitador que es responsable de asegurarse de que el Equipo Scrum se adhiera a los valores y principios de la metodología Agile.

Además, el Scrum Master asegura la adherencia a los procesos y prácticas que el equipo acordó que usarían.

- Propietario del producto: Es el cliente o alguien designado por la organización que conoce sus necesidades. Se encarga de escuchar e integrar las necesidades involucradas en determinado producto, cuidando que los resultados y el valor entregado sean los necesarios.
- Equipo Scrum: Todas aquellas personas que conforman el grupo de trabajo que se encargan de la realización del proyecto.

Ceremonias en Scrum

Scrum se ejecuta en lo que se conoce como sprints o breves iteraciones de trabajo Durando generalmente no más de dos semanas. Un sprint emplea cuatro ceremonias de scrum diferentes para garantizar una ejecución adecuada. Estas ceremonias de scrum se describen a continuación:

- Planificación de Sprint: Aquí es donde el equipo se reúne y decide lo que deben completar en el próximo sprint
- Scrum diario: Esta es una reunión de pie o una muy breve mini reunión de 15 minutos para que el equipo se asegure de que todos estén en la misma página.
- Revisión de Sprint: Este es otro tipo de reunión, pero en la que el equipo demuestra lo que terminado en el sprint.
- Retrospectiva: En esta reunión el equipo revisa su desempeño, identificando lo que hicieron bien y lo que no salió según lo planeado, para que puedan mejorar el próximo sprint.

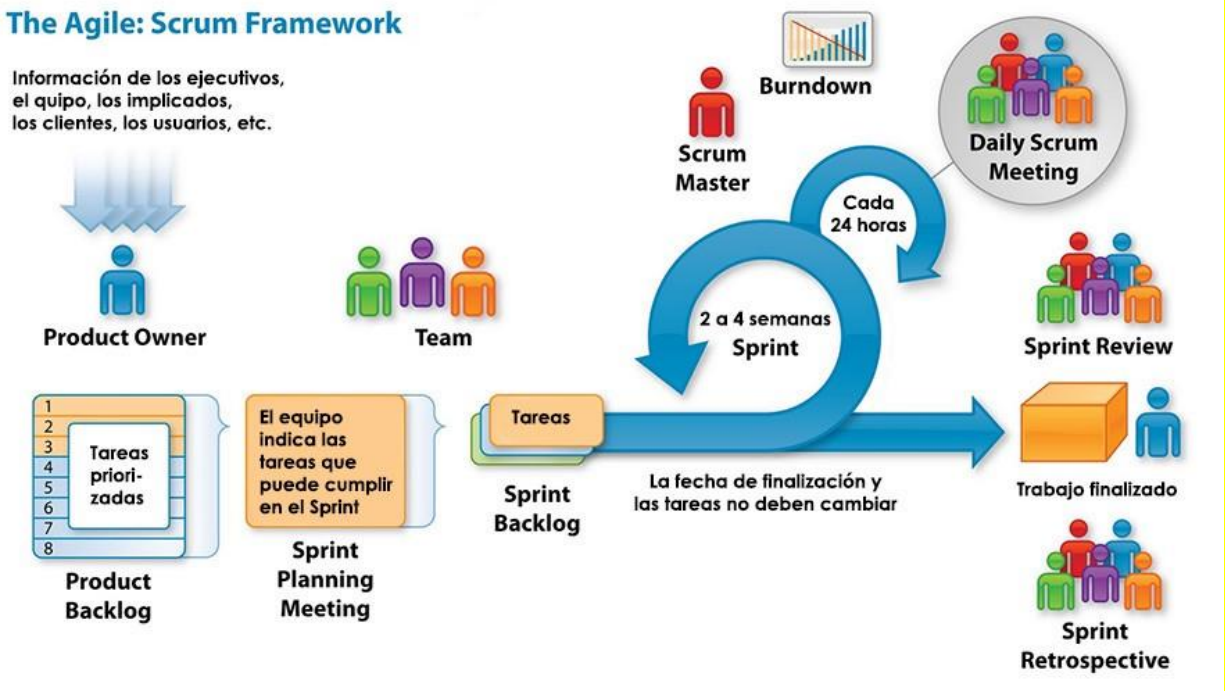
Artefactos en Scrum

Son aquellos que nos van a garantizar la transparencia y organización de las actividades a realizar.

- Lista de producto (Product Backlog): Es una lista ordenada de requerimientos llamados historias de usuario que especifican las necesidades de negocio de forma prioritaria y este es el canal único para incluir más actividades
- Lista de actividades específicas del sprint (Sprint backlog): Es el conjunto de actividades que promete entregar el equipo al finalizar el sprint para garantizar terminar los productos solicitados.
- Incremento: Es la suma de todas las actividades realizadas por el equipo durante ese sprint. Cuando se llega a este punto el usuario final para el que se trabaja podrá tener una interacción y revisar el progreso. Finalmente, aquí se podrá medir la productividad de los equipos de trabajo.

Flujo de trabajo

En el siguiente gráfico se puede ver de forma general el flujo de trabajo en Scrum.

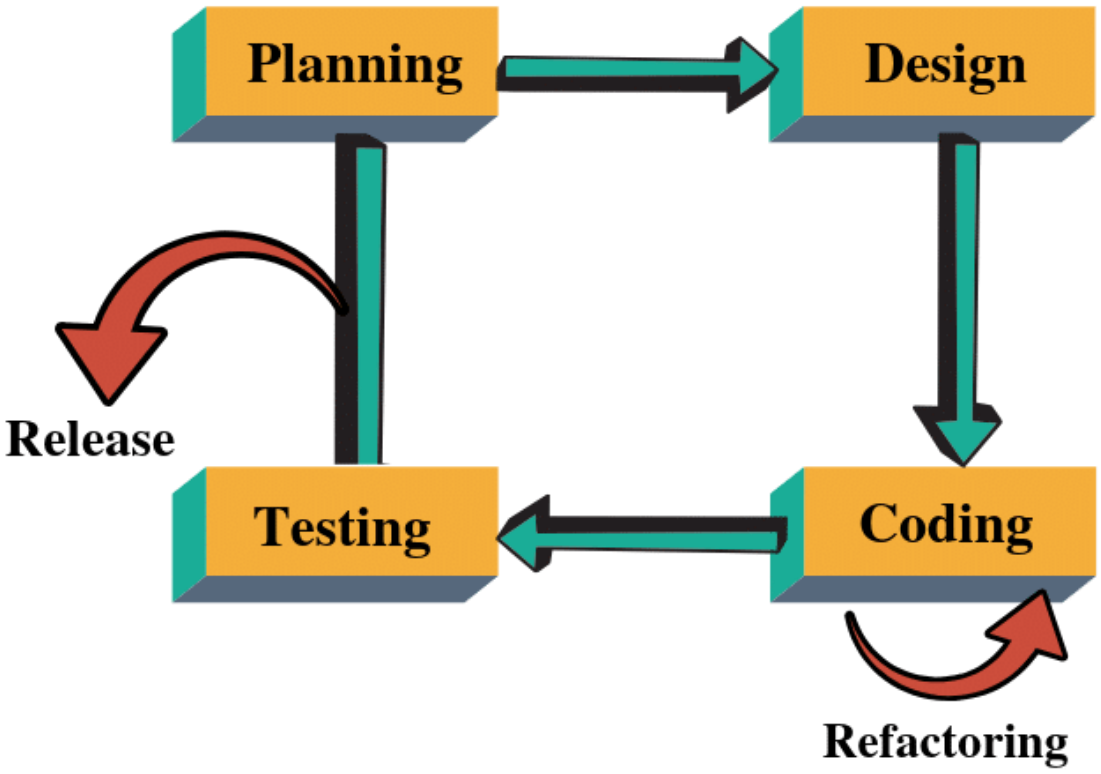


Extreme Programming (XP)

Esta metodología ágil se enfoca en adaptar sobre la marcha el software según las necesidades del cliente. Según esta metodología, los cambios son naturales, y se pueden aplicar en cualquier etapa del proyecto. Una práctica común es la programación en parejas en la cual una persona realiza el trabajo de desarrollo y otra persona revisa en tiempo real esto permite discutir las mejoras que se pueden aplicar y también minimiza los errores. Extreme Programming trata de evitar planificaciones largas y únicamente se planifican las tareas del día. Este tipo de programación es más riesgosa ya que si se comete un error en una etapa avanzada del proyecto, entonces puede significar un gran costo para corregirlo.

Proceso de XP

El proceso de XP es el siguiente:



1. Planificación: Es la primera etapa del proceso es comenzar a recopilar los requisitos de los clientes en forma de historias de usuarios. El valor de cada historia de usuario se estima a continuación, lo que debería ayudar a que el equipo entienda cuánto tiempo se tardará en poner en práctica una historia
2. Diseño: Tener un diseño simple de un sistema asegura menos dependencias y entregas tempranas. La adición de cualquier funcionalidad adicional puede suceder más tarde también. Además, si algunas historias necesitan más investigación en términos de tecnicismo, se asigna un breve período de tiempo llamado Spike.
3. Codificación: Este es el núcleo de cualquier proceso. La codificación real del software comienza aquí. En otras palabras, el equipo se sienta junto y comienza a trabajar gradualmente para entregar el producto final.

4. Prueba: Hacia el final, el código se prueba primero y se ejecuta más tarde, después de corregir los errores.
5. Lanzamiento: Finalmente, se libera el producto valorado por el cliente y se calcula la velocidad del proyecto para mantener una pestaña del progreso. Se mide la velocidad del proyecto, que es un criterio de cuantificación de la cantidad de trabajo realizado en un proyecto.

Bibliografía

- https://www.atlassian.com/blog/software-teams/software-development-trends-2016
- https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software
- https://www.megapractical.com/blog-de-arquitectura-soa-y-desarrollo-de-software/metodologias-de-desarrollo-de-software
- https://www.scrum.org/resources/what-is-scrum
- https://www.researchgate.net/publication/220018244_What_Is_the_Rational_Unified_Process

Descargo de responsabilidad

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educacionales del curso.