UNIVERSIDAD AUTÓNOMA GABRIEL RENÉ MORENO

FACULTAD DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

TAREA # 3 Paradigmas de Desarrollo de Software

NOMBRE: MOGIANO GUTIERREZ MOISES LEONARDO

NRO. REGISTRO: 218034121

CARRERA: ING. DE SISTEMAS

DOCENTE: MSC.ING. ANGÉLICA GARZÓN CUÉLLAR

MATERIA: SISTEMAS DE INFORMACIÓN 1-INF342-SA

FECHA: 28/10/2021

SANTA CRUZ – BOLIVIA

TAREA #3 PARADIGMAS DE DESARROLLO DE SOFTWARE

Responder las siguientes preguntas:

1) **¿Qué es un Ciclo de Vida?**

   Un ciclo de vida, desde un punto de vista biológico, es un período que incluyen todas las diferentes especies que, mediante la reproducción, ya sea a través de la reproducción asexual o sexual, generan organismos idénticos a partir de otros.

2) **¿Qué es un Proceso?**

   Un proceso es una serie de tareas interrelacionadas que, juntas transforman las entradas en salidas.1 Estas pueden ser realizadas por personas, la naturaleza o máquinas utilizando diversos recursos; un proceso de ingeniería debe considerarse en el contexto de los agentes que realizan las tareas y los atributos de recursos involucrados.2 Los documentos normativos de la ingeniería de sistemas y los relacionados con los modelos de madurez se basan generalmente en procesos.

3) **¿Qué es un Producto de Software?**

   Un producto de software es una unidad lógica de compartición y empaquetado de software que tiene un desarrollo gestionado, un ciclo de vida de mantenimiento y atributos visibles para el cliente. Puede ser una colección de componentes, productos de software cuya licencia puede depender de la licencia de la oferta total.

4) **Realizar las actividades que tienen los ciclos de vida de la ingeniería de software?**

   Modelo de cascada
   El modelo de cascada es el modelo de paradigma más simple en desarrollo de software. Sigue un
   modelo en que las fases del SDLC funcionarán una detrás de la otra de forma lineal. Lo que
   significa que solamente cuando la primera fase se termina se puede empezar con la segunda, y
   así progresivamente.

   Modelo repetitivo
   Este modelo guía el proceso de desarrollo de software en repeticiones. Proyecta el proceso de
   desarrollo de forma cíclica repitiendo cada paso después de cada ciclo en el proceso de SDLC.

Modelo en espiral
El modelo en espiral es una combinación de ambos modelos, el repetitivo y uno del modelo SDLC.
Se puede ver como si se combina un modelo de SDLC combinado con un proceso cíclico (modelo
repetitivo).
Modelo V
El mayor inconveniente del modelo de cascada es que solo se pasa a la siguiente fase cuando se
completa la anterior, por tanto, no es posible volver atrás si se encuentra algún error en las
etapas posteriores. El Modelo V aporta opciones de evaluación del software en cada etapa de
manera inversa.

Modelo Big Bang
Este modelo es el modelo con la forma más simple. Requiere poca planificación, mucha
programación y también muchos fondos. Este modelo se conceptualiza alrededor de la teoría de
creación del universo 'Big Bang'. Tal como cuentan los científicos, después del big bang
muchas galaxias, planetas y estrellas evolucionaron. De la misma manera, si reunimos muchos
fondos y programación, quizá podemos conseguir el mejor producto de software.

5) Describa las fases de los ciclos de vida convencionales

Las Fases de desarrollo de software
La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y
administrar un proyecto para llevarlo a cabo con grandes posibilidades de éxito.
Esta sistematización indica cómo se divide un proyecto en módulos más pequeños para
normalizar cómo se administra el mismo.
Así, una metodología para el desarrollo de software son los procesos a seguir sistemáticamente
para idear, implementar y mantener un producto de software desde que surge la necesidad del
producto hasta que se cumple el objetivo por el cual fue creado.
De esta forma, las etapas del desarrollo de software son las siguientes:

Planificación
Antes de empezar un proyecto de desarrollo de un sistema de información, es necesario hacer

ciertas tareas que influirán decisivamente en el éxito del mismo. Dichas tareas son conocidas
como el fuzzy front-end del proyecto, puesto que no están sujetas a plazos.
Algunas de las tareas de esta fase incluyen actividades como la determinación del ámbito del
proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados, la
estimación del coste del proyecto, su planificación temporal y la asignación de recursos a las
diferentes etapas del proyecto.

Análisis
Por supuesto, hay que averiguar qué es exactamente lo que tiene que hacer el software. Por eso,
la etapa de análisis en el ciclo de vida del software corresponde al proceso a través del cual se
intenta descubrir qué es lo que realmente se necesita y se llega a una comprensión adecuada de
los requerimientos del sistema (las características que el sistema debe poseer).

Diseño
En esta fase se estudian posibles opciones de implementación para el software que hay que
construir, así como decidir la estructura general del mismo. El diseño es una etapa compleja y su
proceso debe realizarse de manera iterativa.
Es posible que la solución inicial no sea la más adecuada, por lo que en tal caso hay que refinarla.
No obstante, hay catálogos de patrones de diseño muy útiles que recogen errores que otros han
cometido para no caer en la misma trampa.

Implementación
En esta fase hay que elegir las herramientas adecuadas, un entorno de desarrollo que facilite el
trabajo y un lenguaje de programación apropiado para el tipo de software a construir.
Esta elección dependerá tanto de las decisiones de diseño tomadas como del entorno en el que
el software deba funcionar.
Al programar, hay que intentar que el código no sea indescifrable siguiendo distintas pautas
como las siguientes:
1.Evitar bloques de control no estructurados.
2.Identificar correctamente las variables y su alcance.
3.Elegir algoritmos y estructuras de datos adecuadas para el problema.

4.Mantener la lógica de la aplicación lo más sencilla posible.
5.Documentar y comentar adecuadamente el código de los programas.
6.Facilitar la interpretación visual del código utilizando reglas de formato de código previamente
consensuadas en el equipo de desarrollo.
También hay que tener en cuenta la adquisición de recursos necesarios para que el software
funcione, además de desarrollar casos de prueba para comprobar el funcionamiento del mismo
según se vaya programando.

Pruebas
Como errar es humano, la fase de pruebas del ciclo de vida del software busca detectar los fallos
cometidos en las etapas anteriores para corregirlos. Por supuesto, lo ideal es hacerlo antes de
que el usuario final se los encuentre. Se dice que una prueba es un éxito si se detecta algún error.
Instalación o despliegue
La siguiente fase es poner el software en funcionamiento, por lo que hay que planificar
el entorno teniendo en cuenta las dependencias existentes entre los diferentes componentes del
mismo.
Es posible que haya componentes que funcionen correctamente por separado, pero que
al combinarlos provoquen problemas. Por ello, hay que usar combinaciones conocidas que
no causen problemas de compatibilidad.
Uso y mantenimiento
Esta es una de las fases más importantes del ciclo de vida de desarrollo del software. Puesto que
el software ni se rompe ni se desgasta con el uso, su mantenimiento incluye tres puntos
diferenciados:
-Eliminar los defectos detectados durante su vida útil (mantenimiento correctivo).
-Adaptarlo a nuevas necesidades (mantenimiento adaptativo).
-Añadirle nuevas funcionalidades (mantenimiento perfectivo).
Aunque suene contradictorio, cuanto mejor es el software más tiempo hay que invertir en su
mantenimiento. La principal razón es que se usará más (incluso de formas que no se habían
previsto) y, por ende, habrá más propuestas de mejoras.

6) **Elaborar una lista con su descripción de los productos de Software**

Comix.- Un popular visor de historietas digitales, que permite abrir archivos de imagen de
diversos formatos para tener una experiencia de lectura similar a la de la historieta en físico,
pudiendo determinar el tamaño, el zoom de la imagen, etc.

OneNote. Esta herramienta sirve para tomar y administrar notas personales, como lo haría una
libretita en el bolsillo. Empleándola se tiene acceso rápido a listas, anotaciones o recordatorios,
por lo que hace las veces de agenda también.

MediaMonkey.-Una aplicación que permite reproducir, ordenar y administrar archivos de
música y video, a través de una serie de bibliotecas que atienden a autor, disco, y otras
informaciones relevantes, así como sincronizarlas con dispositivos móviles como reproductores
de música y teléfonos celulares.

Mozilla Firefox.- Uno de los más populares navegadores de Internet, disponible para descarga
gratuita. Permite la interacción del usuario con la World Wide Web, así como realizar búsquedas
de datos y otro tipo de interacciones vitales.

Microsoft Word.-Probablemente el más popular procesador de texto del mundo, parte del
paquete de Microsoft Office, que incluye herramientas para negocios, gestión de bases de datos,
elaboración de presentaciones, etc.

Google Chrome.- El navegador de Google, impuso un paradigma de ligereza y velocidad en el
campo de los exploradores de Internet y se hizo rápidamente popular entre los adeptos a la
Internet. Su éxito fue tal que abrió la puerta a proyectos de sistemas operativos Google y otro
software venidero.

Adobe Photoshop. Aplicación para la edición de imágenes, elaboración de contenido visual de
diseño y diversas labores de retocado fotográfico, composición estética y demás, de la empresa
Adobe Inc. Es sin duda un software popular en el mundo del diseño gráfico.

7) ¿Cuáles son las características de los productos de Software?

Las características deseables en un producto de software son:

-Corrección. Que cumpla con su objetivo.

-Usabilidad. Que sea fácil de aprender.

-Seguridad. Que sea resistente a ataques externo.

-Flexibilidad. Que pueda ser modificado por los desarrolladores.

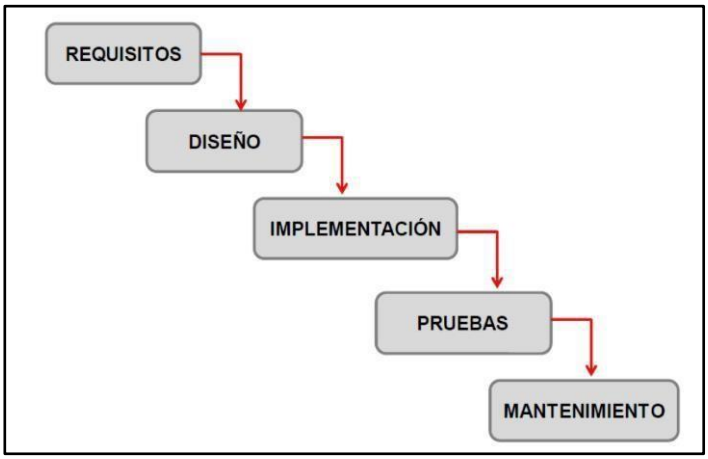
-Portabilidad. Que pueda ser utilizado en diversos equipos.

INVESTIGAR LOS PARADIGMAS DE DESARROLLO DE SOFTWARE

1) CICLO DE VIDA CLASICO (CASCADA)

Es un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.

El modelo en cascada es un proceso de desarrollo secuencial, en el que el desarrollo se ve fluyendo hacia abajo (como una cascada) sobre las fases que componen el ciclo de vida.



1.Especificación de requisitos.

2.Diseño.

3.Construcción (Implementación o codificación).

4.Integración.

5.Pruebas.

6.Instalación.

7.Mantenimiento.
Para seguir el modelo en cascada, se avanza de una fase a la siguiente en una forma puramente secuencial.

2) ESPIRAL

El desarrollo en espiral es un modelo de ciclo de vida desarrollado por Barry Boehm en 1985, utilizado de forma generalizada en la ingeniería del software. Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior.

Al ser un modelo de ciclo de vida orientado a la gestión de riesgos se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente riesgos.

Tareas: Para cada ciclo habrá cuatro actividades:

1. Determinar o fijar objetivos:

• Fijar también los productos definidos a obtener: requerimientos, especificación, manual de usuario.
• Fijar las restricciones.
• Identificar riesgos del proyecto y estrategias alternativas para evitarlos.
• Hay una cosa que solo se hace una vez: planificación inicial o previa

2. Análisis del riesgo:

Estudiar todos los riesgos potenciales y se seleccionan una o varias Alternativas propuestas para reducir o eliminar los riesgos

3. Desarrollar, verificar y validar (probar):

• Tareas de la actividad propia y de prueba.
• Análisis de alternativas e identificación de resolución de riesgos.
• Dependiendo del resultado de la evaluación de riesgos, se elige un modelo para el desarrollo, que puede ser cualquiera de los otros existentes, como formal, evolutivo, cascada, etc. Así, por ejemplo, si los riesgos de la interfaz de usuario son

dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos.

4. Planificar: Revisar todo lo que se ha llevado a cabo, evaluándolo y decidiendo si se continua con las fases siguientes y planificando la próxima actividad.
El proceso empieza en la posición central. Desde allí se mueve en el sentido de las agujas del reloj.

3) TECNICAS DE CUARTA GENERACION

Las técnicas de cuarta generación son un conjunto muy diverso de métodos y herramientas que tienen por objeto el de facilitar el desarrollo del software, facilitan al que desarrolla el software la propiedad de especificar algunas características del mismo a alto nivel, más tarde, la herramienta genera automáticamente el código fuente a partir de esta especificación.

Los tipos más comunes de generadores de código cubren uno o varios de los siguientes aspectos:

Acceso a base de datos: utilizando lenguajes de consulta de alto nivel.

Generadores de códigos: a partir de una especificación de los requisitos se genera automáticamente toda la aplicación

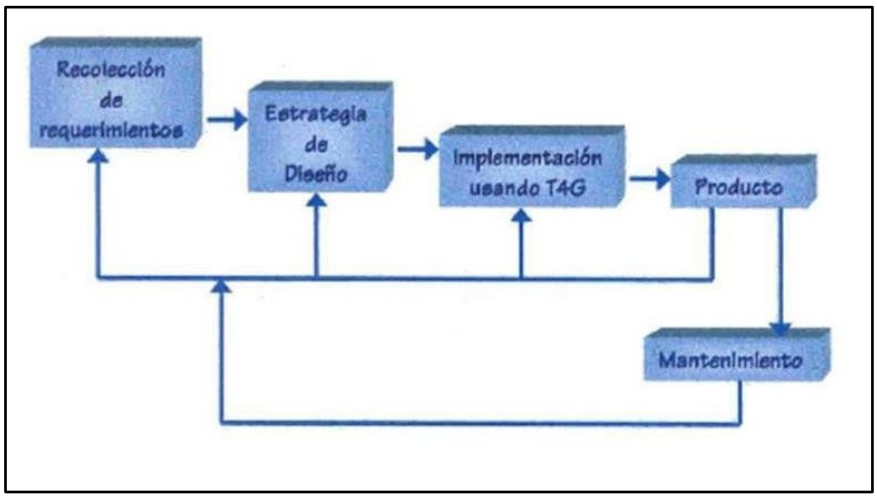
Generación de pantallas: permitiendo diseñar la pantalla dibujándola directamente, incluyendo además el control del cursor y la gestión de los errores de los datos de entrada.
Gestión de entornos gráficos.
Generación de informes.

Como otros paradigmas, T4G comienza con el paso de recolección de requerimientos. En el mejor de los casos el cliente debería describir los requerimientos y éstos traducirse directamente a un prototipo operacional, pero en general esto no es así. El cliente puede no estar seguro de lo que necesita, puede ser ambiguo en la especificación de hechos que son conocidos y puede ser incapaz o no desear especificar la información en la forma que una herramienta T4G puede construirla, además las herramientas actuales T4G no son lo suficientemente sofisticadas para acomodar realmente lenguaje natural y no lo serán por algún tiempo.

Para aplicaciones pequeñas puede ser posible ir directamente desde el paso de establecimiento de requerimientos a la implementación, sin embargo, es necesaria una estrategia del diseño para el sistema. El uso de T4G sin diseño para grandes proyectos causará las mismas dificultades (poca calidad, pobre mantenimiento, mala aceptación por el cliente) que se encuentran cuando se desarrolla software usando los métodos convencionales.

El último paso de la figura anterior contiene la palabra producto para transformar una implementación T4G en un producto, el que lo desarrollo debe dirigir una prueba completa, desarrollar una documentación con sentido y ejecutar todas las otras actividades de transición requeridas en los otros paradigmas de la ingeniería de software.

Los defensores aducen reducciones dramáticas en el tiempo de desarrollo en el software y una mejora significativa en la productividad de la gente que construye el software. Los detractores de este paradigma aducen que los lenguajes de programación, que el código fuente producido por tales herramientas es ineficiente y que el mantenimiento de grandes sistemas de software desarrollado usando T4g está abierta a discusión.



4) PROTOTIPOS

El paradigma de construcción de prototipos comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un diseño rápido. El diseño rápido se centra en una representación de esos aspectos

del software que serán visibles para el usuario/cliente. El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer.

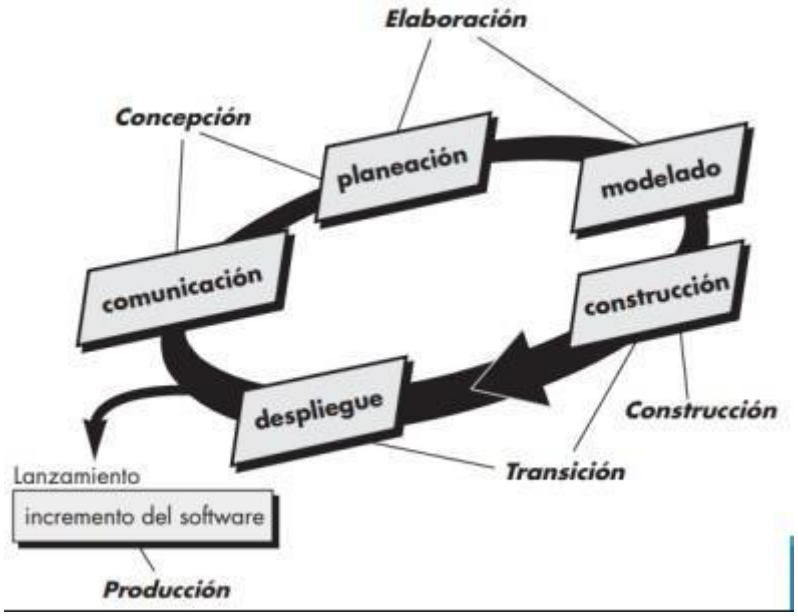
5) PUDS DE PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE

PUDS (Proceso Unificado de Desarrollo de Software) Ivar Jacobson, Grady Booch y James Rumbaugh en su libro Unified Process, analizan la necesidad de un proceso del software "impulsado por el caso de uso, centrado en la arquitectura, iterativo e incremental"
-El proceso unificado es un intento por obtener los mejores rasgos y características de los modelos tradicionales del proceso del software, implementando los mejores principios del desarrollo ágil.

- Reconoce la importancia de la comunicación con el cliente y los métodos directos para describir su punto de vista respecto de un sistema (el caso de uso)

- Ayuda a que el arquitecto se centre en las metas correctas: que sea comprensible, permita cambios futuros y la reutilización (Hace énfasis en la importancia de la arquitectura del software)

- Sugiere un flujo del proceso iterativo e incremental



Fases

-Comunicación con el cliente y planeación.

-Identifica los requerimientos del negocio.

-Propone una arquitectura aproximada para el sistema y se desarrolla un plan para la naturaleza iterativa e incremental del proyecto en cuestión.

6) SCRUM (MANIFIESTO AGIL)

Scrum es el término dado por Nonaka y Takeuchi al método de desarrollo de nuevos productos realizado con equipos reducidos, multidisciplinares, que trabajan con comunicación directa y empleando ingeniería concurrente, en lugar de ciclos o fases secuenciales.

FASES DE SCRUM

SCRUM comprende las siguientes fases:

1.- Pre-juego

Planificación: Definición de una nueva versión basada en la pila actual, junto con una estimación de coste y agenda. Si se trata de un nuevo sistema, esta fase abarca tanto la visión como el análisis. Si se trata de la mejora de un sistema existente comprende un análisis de alcance más limitado. Arquitectura:

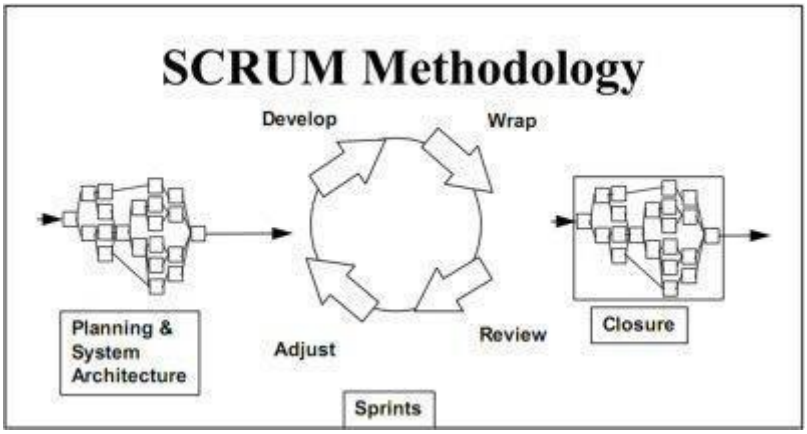
Diseño de la implementación de las funcionalidades de la pila. Esta fase incluye la modificación de la arquitectura y diseño generales.

2.- Juego

Desarrollo de sprints: Desarrollo de la funcionalidad de la nueva versión con respeto continuo a las variables de tiempo, requisitos, costo y competencia. La interacción con estas variables define el final de esta fase. El sistema va evolucionando a través de múltiples iteraciones de desarrollo o sprints.

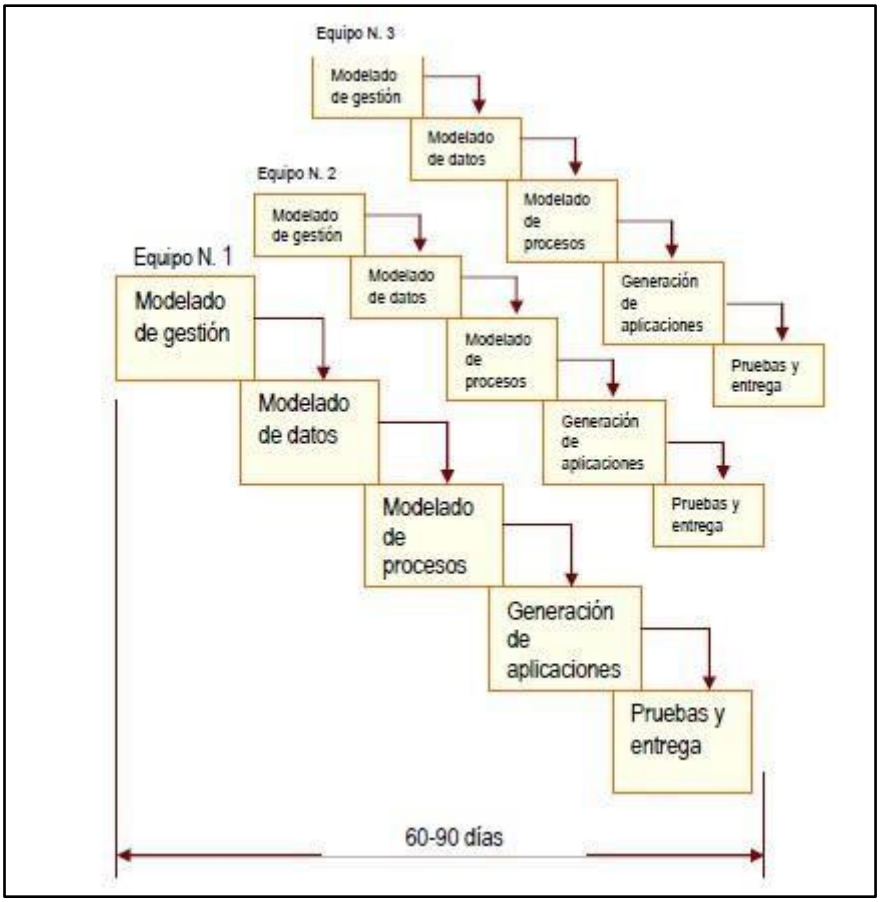
3.- Post-juego

Preparación para el lanzamiento de la versión, incluyendo la documentación final y pruebas antes del lanzamiento de la versión.



7) DRA (DESARROLLO RAPIDO DE APLICACIONES)

Es el proceso de desarrollo de software diseñado para facilitar y acelerar la creación de aplicaciones, que permite construir sistemas utilizables en poco tiempo, normalmente de 60 a 90 días. En conclusión, es una adaptación a "Alta velocidad" en el que se logra el desarrollo rápido utilizando un enfoque de construcción basado en componentes. Si se comprenden bien los requisitos y se limita el ámbito del proyecto, el proceso DRA permite al equipo de desarrollo crear un "sistema completamente funcional" dentro de periodos cortos de tiempo.



CARACTERÍSTICAS DEL MODELO DRA

Según (De Armas Ramírez, (2003)) Afirma que Debido a que el software o aplicación se requiere lo más pronto posible no existe una especificación del sistema detallada.

- El software no se desarrolla y utiliza en su totalidad, sino en una serie de incrementos, donde en cada incremento se incluyen nuevas funcionalidades al sistema.

- A menudo se desarrollan las interfaces de usuario del sistema utilizando un sistema de desarrollo interactivo que permite que el diseño de la interfaz se cree rápidamente dibujando y colando iconos en la interfaz.

- Para su desarrollo se utilizan herramientas de desarrollo visual para agilizar el proceso.

- Se necesitan equipos compuestos por alrededor de seis personas, incluyendo desarrolladores y usuarios de tiempo completo, así como aquellas personas involucradas en los requisitos.

- Las funciones secundarias son eliminadas como sea necesario para cumplir con el calendario.

FASES DEL DRA

Según (March, (2010)) Afirma que Las Fases Son:

Modelado de Gestión
El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce el proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la proceso?

Modelado de Datos

El flujo de información definido como parte de la fase de modelado de gestión se refina como un conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (llamadas atributos) de cada uno de los objetos y las relaciones entre estos objetos.

Modelado de Procesos

Los objetos de datos definidos en la fase de modelado de datos quedan transformados para lograr el flujo de información necesario para implementar una función de gestión. Las descripciones del proceso se crean para añadir, modificar, suprimir, o recuperar un objeto de datos. Es la comunicación entre los objetos.

Generación de Aplicaciones

El DRA asume la utilización de técnicas de cuarta generación. En lugar de crear software con lenguajes de programación de tercera generación, el proceso DRA trabaja para volver a utilizar componentes de programas ya existentes (cuando es posible) o a crear componentes reutilizables (cuando sea necesario).

Pruebas de Entrega

Como el proceso DRA enfatiza la reutilización, ya se han comprobado muchos de los componentes de los programas. Esto reduce tiempo de pruebas.

Sin embargo, se deben probar todos los componentes nuevos y se deben ejercitar todas las interfaces a fondo.

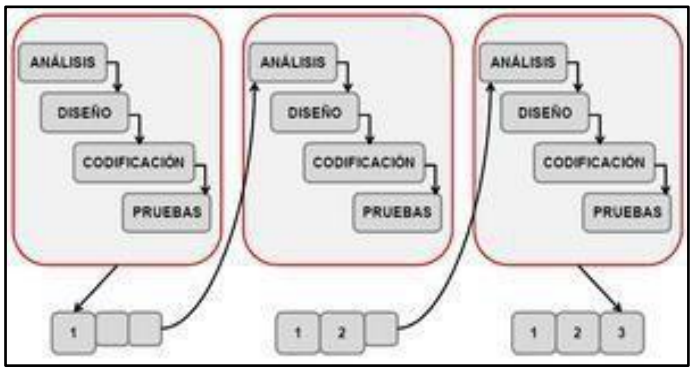
Ventajas y desventajas del "DRA"

Según (Lorences González, (2003)) Afirma Que Las: Los entregables pueden ser fácilmente trasladados a otra plataforma, El desarrollo se realiza a un nivel de abstracción mayor, Entrega temprana al cliente, Compromiso del cliente con el sistema, Mayor flexibilidad, Menor codificación manual, Mayor involucramiento de los usuarios, Posiblemente menos fallas, Posiblemente menor costo, Ciclos de desarrollo más pequeños, Interfaz gráfica estándar.

Y sus Desventajas son: Tiene inconvenientes para proyectos grandes, necesita suficientes recursos humanos para crear el número correcto de equipos, Si los desarrolladores y clientes no se comprenden con las actividades necesarias para completar el sistema, los proyectos fallarán, Un alto costo de herramientas integradas y equipo necesario, Progreso más difícil de medir, Menos eficiente y con menor precisión científica.

8) INCREMENTAL

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.



9) DESARRALLO BASADOS EN COMPONENTES

El desarrollo de software basado en componentes (en lo adelante DSBC) constituye una aproximación del desarrollo de software que describe, construye y emplea técnicas software para elaborar sistemas abiertos y distribuidos, mediante el ensamblaje de partes software reutilizables.

Características y utilidades

Es utilizado para reducir los costos, tiempo y esfuerzos de desarrollo del software, y de esta manera incrementar el nivel de productividad de los grupos desarrolladores y minimizar los riesgos; a su vez ayuda a optimizar la fiabilidad, flexibilidad y la reutilización de la aplicación final. De esta manera, las pequeñas empresas pueden tener una mayor confiabilidad a la hora de realizar una inversión tecnológica.

El modularidad, la reusabilidad y compatibilidad son características muy relevantes de la tecnología de programación basada en componentes, en las cuales coincide con la tecnología orientada a objetos de la que puede considerarse una evolución. No obstante, en esta tecnología también se requiere robustez debido a que los componentes deben operar en entornos muchos más heterogéneos.

El DSBC, se corresponde al paradigma de programación de sistemas abiertos, los cuales son extensibles y tienen una interacción con componentes heterogéneos que se integran o abandonan el sistema de manera dinámica, o sea, que los componentes pueden ser substituidos por otros componentes, independientemente de su arquitectura y desarrollo.

Otro aspecto a tener en cuenta en el DSBC, es el poder integrar lo mejor de las tecnologías para realizar aplicaciones personalizadas, ajustadas a los requisitos de los clientes; lo cual le permite a los desarrolladores y/o a la empresa adquirir las tecnologías que más se adapten a sus particularidades, sin incurrir en gastos de licenciamiento o soporte y actualización de grandes soluciones, aunque muchas de estas tecnologías se encuentran bajo la premisa de Freeware y GNU (General Public License), lo cual añade otra gran ventaja.
Beneficios del DSBC

Este enfoque de desarrollo además de que posibilita alcanzar un alto nivel de reutilización de software, también ofrece otros beneficios que no son menos importantes, como, por ejemplo:

- 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 sus componentes, el desarrollador puede actualizar y/o adicionar componentes según sea requerido, sin afectar otras partes del sistema.

- Mayor calidad. Dado que un componente puede ser construido y luego optimizado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

De la misma manera, el hecho de comprar componentes de terceros en lugar de desarrollarlos, posee algunas ventajas:

- Ciclos de desarrollo más cortos. La adición de una pieza dada de funcionalidad tomará días en lugar de tardar meses o incluso años.

- Mejor ROI. Usando correctamente esta estrategia, el retorno sobre la inversión puede ser más favorable que desarrollando los componentes uno mismo.

- Funcionalidad mejorada. Para usar un componente que contenga una pieza de funcionalidad, solo se necesita entender su naturaleza, más no sus detalles internos. Así, una funcionalidad que sería impráctica de implementar en la empresa, se vuelve ahora completamente asequible.

10)PROGRAMACION EXTREMA (XP)

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP en sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea. A continuación, presentaremos las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

PROCESO XP

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos :

1. El cliente define el valor de negocio a implementar.

2. El programador estima el esfuerzo necesario para su implementación.

3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.

4. El programador construye ese valor de negocio.

5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

El ciclo de vida ideal de XP consiste de seis fases : Exploración, Planificación de la Entrega
(Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Fase I: Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las

herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

Fase II: Planificación de la Entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

Fase III: Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué

historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores. Wake en proporciona algunas guías útiles para realizar la planificación de la entrega y de cada iteración.

Fase IV: Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

Fase V: Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

Fase VI: Muerte del Proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios

esperados por el cliente o cuando no hay presupuesto para mantenerlo.

11)WIN WIN

Originalmente el modelo espiral surge para sustituir la solución en fases del "modelo en cascada" con ciclos de experimentación y aprendizaje. El modelo incorpora un nuevo elemento en el desarrollo al cual denomina "análisis de riesgos". El modelo es la unión del modelo de prototipos y la secuencialidad del clásico, esta mezcla permite construir un ciclo que inicia desde el centro, si el cliente prefiere hacer alguna modificación o mejoría al producto, se evalúan las diferentes alternativas y se crea un nuevo análisis de riesgos, con lo cual se forma una nueva vuelta de espiral. De esta manera se incrementa el número de vueltas hasta que el producto queda terminado. Con el tiempo este modelo fue evolucionando en distintas propuestas como el modelo espiral Victoria-Victoria (WinWin). Este nuevo modelo incorpora unas actividades para la comunicación con el cliente para lograr mejores negociaciones que beneficien tanto
al desarrollador como al cliente.

Etapas, fases y/o procesos que propone la misma.

El modelo Espiral Win-Win se basa en la inclusión de tres etapas o regiones al principio:

1.- Identificar las partes interesadas (stakeholders) para esta nueva iteración del producto: Es necesario definir los interlocutores que serán de áreas que se verán afectadas por el resultado final de la

nueva versión. Estos interlocutores serán del área del cliente (puede haber más de uno) y del proveedor.

2.- Identificar las condiciones de victoria de las partes interesadas en el proyecto: Se concreta cuáles son las condiciones que requiere cada parte para que se sienta satisfecha una vez realizada esta versión.

3a.- Reunir las condiciones de victoria: Con las etapas anteriores se han definido unos objetivos generales para la versión y se obtiene conocimiento de los objetivos particulares de cada parte. Ahora toca negociar hasta dónde realmente se va a llegar y cómo, intentando llegar a una solución en la que todos ganen (cliente y proveedor).

Las siguientes etapas tienen una correspondencia con algunas variantes, con la versión inicial del ciclo de vida en espiral:

3b.- Establecer los objetivos, restricciones y alternativas del siguiente nivel: Teniendo en cuenta los objetivos y acuerdos de las fases anteriores, se definirían los requisitos de esta versión, además de especificarse diversas alternativas en el caso de que existan.

4.- Evaluar las alternativas del producto y del proceso y resolución de riesgos: Se realizaría el análisis del riesgo típico de los modelos en espiral.

5.- Definir el siguiente nivel del producto y del proceso, incluyendo particiones: Esta etapa completaría el proceso de análisis y realizaría el diseño y la construcción.

6.- Validar las definiciones del producto y del proceso: Comprendería la implantación y aceptación de la versión, verificándose si el resultado de la iteración satisface el alcance indicado.

7.- Revisión y comentarios: Tocaría hacer inventario, medir el nivel de satisfacción de las partes, el nivel de cumplimiento de objetivos con el objetivo sobre todo de intentar aprender de los errores para mejorar en versiones sucesivas y de detectar correcciones y mejoras a realizar en el producto.

Ventajas y Desventajas

VENTAJAS

1. El modelo en espiral puede adaptarse y aplicarse a lo largo de la vida del software de computadora.

2. Como el software evoluciona a medida que progresa el proceso, el desarrollador y el cliente comprenden y reaccionan mejor ante riesgos en cada uno de los niveles evolutivos.

3. El modelo en espiral permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.

4. El modelo en espiral demanda una consideración directa de los riesgos técnicos en todas las etapas del proyecto y si se aplica adecuadamente debe reducir los riesgos antes de que se conviertan en problemas

DESVENTAJAS

1. Resulta difícil convencer a grandes clientes de que el enfoque evolutivo es controlable.

2. Debido a su elevada complejidad no se aconseja utilizarlo en pequeños sistemas.

3. Genera mucho tiempo en el desarrollo de sistemas.

4. Requiere experiencia en la identificación de riesgos.

12)RECURSIVO PARALELO

En paradigma de programación es una colección de modelos conceptuales que juntos modelan el proceso de diseño y determinan la estructura de un programa.

Tipos de Paradigmas de Programación

a. Que soportan técnicas de programación de bajo nivel

b. Que soportan métodos de diseño de algoritmos

c. Que soportan soluciones de programación de alto nivel

d. Basado para el desarrollo de sistemas expertos

e. De programación lógica

f. De programación funcional

g. De programación heurística

h. Orientado al objeto

Diferentes lenguajes de programación que soportan cada una de estas categorías de paradigmas :

a. Solución procedimental u operacional

b. Solución demostrativa

c. Solución declarativa

13)FRAMEWORK

El propósito de un framework es mejorar la eficiencia de la creación de un nuevo software. Los framework pueden mejorar la productividad del desarrollador y mejorar la calidad, la fiabilidad y robustez del nuevo software. La productividad del desarrollador se ha mejorado al permitir a los desarrolladores centrarse en las necesidades únicas de su aplicación, en lugar de pasar tiempo en la infraestructura de aplicaciones.

Hay varias características que diferencia un FrameWork y una librería o aplicación.

1. El Framework tiene un comportamiento predeterminado.

2. Un Framework puede ser ampliado por el usuario proporcionando una funcionalidad específica.

3. El código del Framework no puede ser modificado

4. A diferencia de una biblioteca o aplicaciones de usuarios el flujo del programa no es controlado por la persona sino por el FrameWork.

Los Frameworks se basan en el Modelo Vista Controlador (MVC), un patrón de diseño que separa las aplicaciones en tres componentes:

*__Modelo:__ son los datos o la información que se manejan en la aplicación.

*__Vista:__ normalmente representada por una interfaz de usuario, presenta el modelo en un formato elegido.

*__Controlador:__ es la capa intermedia. Se encarga de gestionar las peticiones recibidas desde la vista, interactuando con la capa de modelo.

Ventajas:

*Un framework facilita el desarrollo de software permitiendo a los diseñadores y programadores dedicar su tiempo a lograr los requerimientos de software en lugar de lidiar con los detalles de bajo nivel necesarios para obtener un sistema funcional, de esta forma se puede reducir el tiempo total de desarrollo de la aplicación.

*Utilizan patrones de diseño, lo cual permite que el código resultante sea limpio y extensible para futuras ampliaciones. Entre ellos destaca el Modelo Vista Controlador que comentamos anteriormente.

*Facilitan servicios genéricos necesarios en la mayoría de proyectos. De esta forma, también se utiliza código ya testeado, evitando así en el futuro la repetición de errores.

*Favorecen la reutilización de código, simplificando el proceso de desarrollo. Ello provoca una amplia ganancia de tiempo en cuanto a la programación y el diseño.

*Aumenta la facilidad de depuración del código gracias al MVC.
Desventajas:

*Posibilidad de generación de código innecesario para nuestra aplicación, ya que los Frameworks tienden a generalizar la funcionalidad de los componentes, provocando una demanda de recursos computacionales innecesaria.

***Aprendizaje costoso.** El tiempo que se gana en dejar de programar puede perderse en aprender el Frameworks si no se va a utilizar para otros proyectos.
***Existe una alta dependencia del código fuente de la aplicación con respecto al Framework.** Además, cada Frameworks tiene su propia convención de código, por lo que no resulta sencillo cambiar de Frameworks.

***Si una librería falla, la depuración es más complicada al no conocer el programador el código.** Por eso es importante utilizar Frameworks y módulos en versiones avanzadas.