**OVI PARA DISEÑAR UN OBJETO VIRTUAL DE INFORMACIÓN – OVI**

**INTRODUCCION AL DESARROLLO DE SOFTWARE**

“ESTAPAS DEL DESARROLLO DE SOFTWARE”

**Objetivo General**

El objetivo que tiene el desarrollo de este producto, será el de orientar de manera inicial, a aquellas personas que quieran entrar al maravilloso mundo de la ingeniería de software y sus múltiples beneficios.

No voy a enfocarme en lo complejo o sencillo que pueda ser aprender de esta herramienta, si no a incentivarlos a no Encallar en una isla donde no tengamos recursos de superar aquellas piedras en el camino, y que por el contrario expandamos nuestros conocimientos en el descubrimiento de nuevas herramientas, siempre en pro de un futuro mejor.

**Contenido informativo de la presentación en PPT para diseñar el OVI**

|  |  |
| --- | --- |
| Sección de texto ( Conociendo el Sistema de Software) | Sección de imágenes |
| **INTRODUCCIÓN**  Un sistema informático está compuesto por hardware y software. En cuanto al hardware, su producción se realiza sistemáticamente y la base de conocimiento para el desarrollo de dicha actividad está claramente definida. La fiabilidad del hardware es, en principio, equiparable a la de cualquier otra máquina construida por el hombre. Sin embargo, respecto del software, su construcción y resultados han sido históricamente cuestionados debido a los problemas asociados, entre ellos podemos destacar los siguientes [1]:   * Los sistemas no responden a las expectativas de los usuarios. * Los programas “fallan” con cierta frecuencia. * Los costes del software son difíciles de prever y normalmente superan las estimaciones. * La modificación del software es una tarea difícil y costosa. * El software se suele presentar fuera del plazo establecido y con menos prestaciones de las consideradas inicialmente. * Normalmente, es difícil cambiar de entorno hardware usando el mismo software.   El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, etc.) no suele cumplirse.  Pressman caracteriza la Ingeniería de Software como “una tecnología multicapa”, ilustrada en la Figura.  Imagen 1 El proceso de desarrollo del software Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que reúna los requisitos del cliente.  Este proceso es intensamente intelectual, afectado por la creatividad y juicio de las personas involucradas [4]. Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales, relativos esencialmente a la naturaleza del producto obtenido.  A continuación se relacionan algunas particularidades asociadas al desarrollo de software y que influyen en su proceso de construcción   * Un producto software en sí es complejo, es prácticamente inviable conseguir un 100% de confiabilidad de un programa por pequeño que sea. Existe una inmensa combinación de factores que impiden una verificación exhaustiva de las todas posibles situaciones de ejecución que se puedan presentar (entradas, valores de variables, datos almacenados, software del sistema, otras aplicaciones que intervienen, el hardware sobre el cual se ejecuta, etc.). * Un producto software es intangible y por lo general muy abstracto, esto dificulta la definición del producto y sus requisitos, sobre todo cuando no se tiene precedentes en productos software similar. Esto hace que los requisitos sean difíciles de consolidar tempranamente. Así, los cambios en los requisitos son inevitables, no   **Imagen 2**  El proceso de desarrollo de software no es único. No existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo. Debido a esta diversidad, es difícil automatizar todo un proceso de desarrollo de software.  A pesar de la variedad de propuestas de proceso de software, existe un conjunto de actividades fundamentales que se encuentran presentes en todos ellos:   1. Especificación de software: Se debe definir la funcionalidad y restricciones operacionales que debe cumplir el software. 2. Diseño e Implementación: Se diseña y construye el software de acuerdo a la especificación. 3. Validación: El software debe validarse, para asegurar que cumpla con lo que quiere el cliente. 4. Evolución: El software debe evolucionar, para adaptarse a las necesidades del cliente.   **proceso de desarrollo de software Según Pressman:**   * **Un marco común del proceso**, definiendo un pequeño número de actividades del marco de trabajo que son aplicables a todos los proyectos de software, con independencia del tamaño o complejidad. * **Un conjunto de tareas**, cada uno es una colección de tareas de ingeniería del software, hitos de proyectos, entregas y productos de trabajo del software, y puntos de garantía de calidad, que permiten que las actividades del marco de trabajo se adapten a las características del proyecto de software y los requisitos del equipo del proyecto. * **Las actividades de protección**, tales como garantía de calidad del software, gestión de configuración del software y medición, abarcan el modelo del proceso. Las actividades de protección son independientes de cualquier actividad del marco de trabajo y aparecen durante todo el proceso.   **Imagen 3** | Imagen 1  <https://upload.wikimedia.org/wikipedia/commons/thumb/2/2e/El_modelo_de_desarrollo_en_cascada.svg/350px-El_modelo_de_desarrollo_en_cascada.svg.png>    Imagen 2    Imagen 3  <http://www.nosolousabilidad.com/articulos/img/uxd6.png>  <https://www.ecured.cu/images/7/71/Pds1.JPG> |

|  |  |
| --- | --- |
| Sección de texto (Introducción al Desarrollo de Software) | Sección de imágenes |
| Modelos de proceso software Sommerville define modelo de proceso de software como “Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real.”  Los modelos genéricos no son descripciones definitivas de procesos de software; sin embargo, son abstracciones útiles que pueden ser utilizadas para explicar diferentes enfoques del desarrollo de software.  Modelos que se van a discutir a continuación:   * Codificar y corregir * Modelo en cascada * Desarrollo evolutivo * Desarrollo formal de sistemas * Desarrollo basado en reutilización * Desarrollo incremental * Desarrollo en espiral  Codificar y corregir (Code-and-Fix) Este es el modelo básico utilizado en los inicios del desarrollo de software. Contiene dos pasos:   * Escribir código. * Corregir problemas en el código.   Se trata de primero implementar algo de código y luego pensar acerca de requisitos, diseño, validación, y mantenimiento.  Este modelo tiene tres problemas principales [7]:   * Después de un número de correcciones, el código puede tener una muy mala estructura, hace que los arreglos sean muy costosos. * Frecuentemente, aún el software bien diseñado, no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara. * El código es difícil de reparar por su pobre preparación para probar y modificar.  Modelo en cascada El primer modelo de desarrollo de software que se publicó se derivó de otros procesos de ingeniería [8]. Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.  El modelo en cascada consta de las siguientes fases:   1. Definición de los requisitos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema. Se busca hacer esta definición en detalle. 2. Diseño de software: Se particiona el sistema en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema. 3. Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad. 4. Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente. 5. Operación y mantenimiento: Generalmente es la fase más larga. El sistema es puesto en marcha y se realiza la corrección de errores descubiertos. Se realizan mejoras de implementación. Se identifican nuevos requisitos.   La interacción entre fases puede observarse en la Figura 5. Cada fase tiene como resultado documentos que deben ser aprobados por el usuario.  Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.  Imagen1 Desarrollo evolutivo La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado. En la Figura 6 se observa cómo las actividades concurrentes: especificación, desarrollo y validación, se realizan durante el desarrollo de las versiones hasta llegar al producto final.  Una ventaja de este modelo es que se obtiene una rápida realimentación del usuario, ya que las actividades de especificación, desarrollo y pruebas se ejecutan en cada iteración. Desarrollo formal de sistemas Este modelo se basa en transformaciones formales de los requisitos hasta llegar a un programa ejecutable.  Recuperado de : <https://www.google.com.co/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0ahUKEwish6H7mMPSAhUFRSYKHXPjC2IQFggrMAM&url=http%3A%2F%2Fwww.dsic.upv.es%2Fasignaturas%2Ffacultad%2Flsi%2Fdoc%2FIntroduccionProcesoSW.doc&usg=AFQjCNGAyOzvxjChM6izsIaI_eLoQhtk2Q&sig2=tl0xIddtWN3DqzRs6BIHWQ&bvm=bv.148747831,d.eWE&cad=rja> | <http://3.bp.blogspot.com/-fSf5HMEIBNw/VdPzH5XGSGI/AAAAAAAAADU/Ikufr9wtX-Q/s1600/INTRODUCCION-AL-DESARROLLO.jpg>    Imagen1  balzer  Imagen 2  <http://3.bp.blogspot.com/_ltFmdWrPtco/TKVUI2GDsfI/AAAAAAAAAAk/1qGBKVUYky0/s1600/imagev.gif>  <http://1.bp.blogspot.com/-0wz1Z9otpAw/USLGQu3MtnI/AAAAAAAAADE/qb8H-pKJX84/s1600/soft12.jpg> |

|  |  |
| --- | --- |
| Sección de texto (DESARROLLO DE SOFTWARE) | Sección de imágenes |
| Desarrollar un [software](https://www.ecured.cu/Software) significa construirlo simplemente mediante su descripción. Esta es una muy buena razón para considerar la actividad de desarrollo de software como una [ingeniería](https://www.ecured.cu/Ingenier%C3%ADa_de_Software). En un nivel más general, la relación existente entre un software y su entorno es clara ya que el software es introducido en el mundo de modo de provocar ciertos efectos en el mismo.  Aquellas partes del mundo que afectarán al software y que serán afectadas por él será el [Dominio de Aplicación](https://www.ecured.cu/index.php?title=Dominio_de_Aplicaci%C3%B3n&action=edit&redlink=1). Es allí donde los [usuarios](https://www.ecured.cu/Usuarios) o [clientes](https://www.ecured.cu/Clientes) observarán si el desarrollo del software ha cumplido su propósito.  Una de las mayores deficiencias en la práctica de construcción de software es la poca atención que se presta a la discusión del problema. En general los desarrolladores se centran en la solución dejando el problema inexplorado. El problema a resolver debe ser deducido a partir de su solución.  Esta aproximación orientada a la solución puede funcionar en campos donde todos los problemas son bien conocidos, clasificados e investigados, donde la innovación se ve en la detección de nuevas soluciones a viejos problemas.  Pero el desarrollo de software no es un campo con tales características. La versatilidad de las [computadoras](https://www.ecured.cu/Computadora) y su rápida evolución hace que exista un repertorio de problemas en constante cambio y cuya solución software sea de enorme importancia.  **DESARROLLO DEL SOFTWARE**  Cuando se va desarrollar un software intervienen muchas personas como lo es el cliente que es el que tiene el problema en su [empresa](https://www.ecured.cu/Empresa) y desea que sea solucionado, para esto existe el [Analista de Sistema](https://www.ecured.cu/index.php?title=Analista_de_Sistema&action=edit&redlink=1) que es el encargado de hacerle llegar todos los requerimientos y necesidades que tiene el cliente a los programadores que son las personas encargadas de realizar lo que es la codificación y [diseño](https://www.ecured.cu/Dise%C3%B1o) del sistema para después probarlo y lo instalan al cliente. Es así como intervienen varias personas ya que una sola persona no podría determinar todo lo necesario lo más seguro que le haga falta algún requerimiento o alguna parte del nuevo sistema y entre más estén involucradas mejor para cubrir con todos los requerimientos del sistema.  **FASES DEL PROCESO DE DESARROLLO DE SOFTWARE**  <https://www.ecured.cu/images/thumb/8/89/Esquema_desarrollo_software.jpg/200px-Esquema_desarrollo_software.jpg>  Extraer los requisitos de un [producto](https://www.ecured.cu/Producto) de software es la primera etapa para crearlo. Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere de [habilidad](https://www.ecured.cu/Habilidad) y experiencia en la ingeniería de software para reconocer requisitos incompletos, ambiguos o contradictorios. El resultado del [análisis](https://www.ecured.cu/An%C3%A1lisis) de requisitos con el cliente se plasma en el documento ERS, Especificación de Requerimientos del Sistema, cuya estructura puede venir definida por varios estándares, tales como CMM-I. Asimismo, se define un [diagrama](https://www.ecured.cu/Diagrama) de Entidad/Relación, en el que se plasman las principales entidades que participarán en el desarrollo del software. La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales. Se han ideado modelos y diversos procesos de trabajo para estos fines. Aunque aún no está formalizada, ya se habla de la Ingeniería de Requisitos. La IEEE Std. 830-1998 normaliza la creación de las Especificaciones de Requisitos Software (Software Requirements Specification).  **DISEÑO Y ARQUITECTURA**  Se refiere a determinar cómo funcionará de forma general sin entrar en detalles. Consiste en incorporar consideraciones de la [implementación tecnológica](https://www.ecured.cu/index.php?title=Implementaci%C3%B3n_tecnol%C3%B3gica&action=edit&redlink=1), como el [hardware](https://www.ecured.cu/Hardware), la [red](https://www.ecured.cu/Red), etc. Se definen los casos de uso para cubrir las funciones que realizará el sistema, y se transforman las entidades definidas en el análisis de requisitos en clases de diseño, obteniendo un modelo cercano a la [programación orientada a objetos](https://www.ecured.cu/Programaci%C3%B3n_orientada_a_objetos).  Programación  Reducir un diseño a código puede ser la parte más obvia del [trabajo](https://www.ecured.cu/Trabajo) de ingeniería de software, pero no es necesariamente la porción más larga. La complejidad y la duración de esta etapa está íntimamente ligada al o a los [lenguajes de programación](https://www.ecured.cu/Lenguaje_de_Programaci%C3%B3n) utilizados.  Pruebas  Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación. Una [técnica](https://www.ecured.cu/T%C3%A9cnica) de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral, para así llegar al objetivo. Se considera una buena práctica el que las pruebas sean efectuadas por alguien distinto al desarrollador que la programó, idealmente un [área](https://www.ecured.cu/%C3%81rea) de pruebas; sin perjuicio de lo anterior el programador debe hacer sus propias pruebas. En general hay dos grandes formas de organizar un área de pruebas, la primera es que esté compuesta por personal inexperto y que desconozca el tema de pruebas, de esta forma se evalúa que la documentación]entregada sea de calidad, que los procesos descritos son tan claros que cualquiera puede entenderlos y el software hace las cosas tal y como están descritas. El segundo enfoque es tener un área de pruebas conformada por programadores con experiencia, personas que saben sin mayores indicaciones en qué condiciones puede fallar una aplicación y que pueden poner atención en detalles que personal inexperto no consideraría.  **DOCUMENTACIÓN**  Todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del [proyecto](https://www.ecured.cu/Proyecto), pasando por modelaciones (UML), [diagramas](https://www.ecured.cu/Diagramas), pruebas, [manuales de usuario](https://www.ecured.cu/Manual_de_usuario), manuales técnicos, etc; todo con el propósito de eventuales correcciones, [usabilidad](https://www.ecured.cu/Usabilidad), mantenimiento futuro y ampliaciones al sistema.  **MANTENIMIENTO**  Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 de toda la ingeniería de software tiene que ver con dar mantenimiento. Una pequeña parte de este trabajo consiste en arreglar errores, o [bugs](https://www.ecured.cu/Bugs). La mayor parte consiste en extender el sistema para hacer nuevas cosas. De manera similar, alrededor de 2/3 de toda la [Ingeniería civil](https://www.ecured.cu/Ingenier%C3%ADa_Civil), [Arquitectura](https://www.ecured.cu/Arquitectura) y trabajo de construcción es dar mantenimiento.  Se puede decir que con la mejora continua garantiza la [calidad](https://www.ecured.cu/Calidad) del [producto](https://www.ecured.cu/Producto), ya que el estarla aplicando día con día es la mejor decisión que puede llegar a tener cualquier empresa, porque de esta manera evita grandes problemas en la elaboración o desarrollo de los productos. Esto es fundamental para todas las empresas ya que se vuelven competitivas, con mayor productividad y eficiencia. No hay que olvidar que la mejora se da porque el cliente es el rey y hay que satisfacer todas y cada una de sus necesidades siempre garantizando la [calidad](https://www.ecured.cu/Calidad).  **Metodología**  Todo desarrollo de software es riesgoso y difícil de controlar, pero si no llevamos una [metodología](https://www.ecured.cu/Metodolog%C3%ADa) de por medio, se obtiene clientes insatisfechos con el resultado y desarrolladores aún más.  Sin embargo muchas veces no se toma en cuenta el utilizar una metodología adecuada, sobre todo cuando se trata de proyectos pequeños de dos o tres meses.  Con relación a los proyectos que se desarrollan con mayor envergadura, hay si se toma el sentido de basarse en una metodología de desarrollo y se empieza a buscar cuál sería la más apropiada para dicho caso. A fin de cuenta no encontramos muchas veces la más adecuada y se termina por hacer un diseño propio de metodología, por supuesto no está mal siempre y cuando sirva para alcanzar el objetivo.  Muchas veces se realiza el diseño del software de manera rígida, tal como el cliente lo solicitó, de esa manera cuando el cliente en la "etapa de prueba" solicita un cambio se hace muy difícil de realizarlo, pues si se hace altera las cosas que no se habían previsto, y este es uno de los factores que atrasan el proyecto y crea incomodidad al desarrollador y en muchas oportunidades no llegan a cumplir con el cambio solicitado, esto conlleva malestar en el cliente puesto que no ha sido tomado en cuenta su pedido; para evitar estos incidentes se debe llegar a un acuerdo formal con el cliente al inicio del proyecto de manera que no perjudique el desarrollo del mismo.  Muchas veces los usuarios finales se dan cuenta que dejaron de mencionar algunas cosas y lo manifiestan en la etapa inicial del proyecto cuando se le muestra el [prototipo](https://www.ecured.cu/Prototipo) del mismo.  **ALGUNAS METODOLOGÍAS CONOCIDAS:**   * La metodología [RUP](https://www.ecured.cu/RUP) es la más adaptable para proyectos de largo plazo. * La metodología [XP](https://www.ecured.cu/XP) en cambio, se recomienda para proyectos de corto plazo. * La metodología [MSF](https://www.ecured.cu/index.php?title=MSF&action=edit&redlink=1) se adapta a proyectos de cualquier dimensión y de cualquier tecnología.   Se puede decir además que lo más importante antes de elegir la metodología que se debe usar para implementar el software, es determinar el alcance que tendrá y luego de allí ver cuál es la que más se acomoda a la aplicación.  **Ejemplos:**  El ejemplo del software lo hacen numerosas empresas, cada vez más gobiernos (registro gratis). Los expertos lo recomiendan, lo hacen particulares a millones. Hasta (a regañadientes) [Microsoft](https://www.ecured.cu/Microsoft). La idea absurda de dejar abierta las tripas del software y permitir que la gente las mire, e incluso que las modifique, copie y use en condiciones diferentes, en la industria de la informática es muy común. De hecho se extiende a los más pequeños rincones del mundo desde una orden mágica hermética de tradición masónica y rosacruciana a telefónica I+D. Si hasta las empresas en filosofía más expuestas o menos rápidas en novación y las sociedades secretas son capaces de ver las ventajas del "[Open Source](https://www.ecured.cu/Open_Source)" (abierto o libre).  No ha sido sencillo la idea conocida como dicho software (abierto o libre) ha tenido una vida larga pero difícil, dirigida por polémicas aparentemente absurdas pero que contienen un profundo debate ideológico y práctico; a veces dividido en partes enfrentadas con mucha pasión; siempre descalificada, lo cierto es que la comunidad del software abierto hoy es una fuerte y sana realidad.  Importancia  Actualmente la transición que estamos viviendo hacia una sociedad del conocimiento ha cambiado profundamente las relaciones entre las personas, empresas y gobiernos: las empresas usan la red para comunicarse con los clientes, utilizan también herramientas de gestión del conocimiento para hacer más eficientes, los gobiernos mejoran su presencia en [Internet](https://www.ecured.cu/Internet) y los servicios a los ciudadanos a través de la red, los usuarios usan las herramientas para sus relaciones personales, etc. Se va de forma imparable hacia una sociedad altamente interconectada donde el eje fundamental es la información.  El software es el intermediario cada vez más grande entre la [información](https://www.ecured.cu/Informaci%C3%B3n) y la [inteligencia](https://www.ecured.cu/Inteligencia) humana. De la misma manera que preocupa para poder acceder a la información, si existe la censura, es tema de preocupación de quien controla este intermediario y las garantías de su transparencia y confiabilidad.  En principio, el software es un [programa informático](https://www.ecured.cu/Programa_inform%C3%A1tico) o conjunto de ellos que tiene un fin determinado, es el de procesar los textos que usamos, el [controlador](https://www.ecured.cu/Controlador) de grabación de nuestros espacios favoritos o las aplicaciones que permiten operar un [teléfono](https://www.ecured.cu/Tel%C3%A9fono) móvil.  Está compuesto por un conjunto de instrucciones que el usuario realiza para ejecutar una función específica. Normalmente los programadores escriben en un lenguaje en el que todos pueden entender y que después es traducido al [lenguaje binario](https://www.ecured.cu/index.php?title=Lenguaje_binario&action=edit&redlink=1) el único que las [máquinas](https://www.ecured.cu/M%C3%A1quinas) entienden. El conjunto de órdenes en el lenguaje que todos trabajan se llaman código fuente.  Si no se accede al código solo se puede usar el programa, no se puede ver cómo está hecho o introducir comentarios. Un ejemplo muy utilizado es el de la receta de cocina, en el que el código fuente son las instrucciones que permite confeccionar un plato. Sin la receta solo se pude degustar el plato, pero no se sabe si se le añade algo vaya en contra de algunos de esos ingredientes ya que se desconocen su composición y proporción. En este sentido, el código fuente juega un papel fundamental en la manera como se debe entender el software.  Se podrían poner varios ejemplos para entender dicha importancia. A finales de los [90](https://www.ecured.cu/1990) se pudo ver en todo el mundo la preocupación por parte de empresa y gobiernos por las consecuencias que podían tener el llamado efecto [2000](https://www.ecured.cu/2000). El famoso error informático era debido al hecho de que muchos programas almacenaban la parte de la fecha correspondiente al año utilizando únicamente dos dígitos, de tal manera, que después del año [99](https://www.ecured.cu/1999) (el [1999](https://www.ecured.cu/1999)) podíamos pasar al año [00](https://www.ecured.cu/2000) (¿ año 2000 o año [1900](https://www.ecured.cu/1900)?) causando todo tipo de errores en el cálculo de período de tiempo.  Los ordenadores de las empresas eléctricas, centrales nucleares, sistema de control de [aviación](https://www.ecured.cu/Aviaci%C3%B3n), bancos y en general, todo el software de uso cotidiano, tuvieron que ser revisados. Finalmente algunas aplicaciones fueron corregidas, otras ya funcionaban correctamente y no hubo que lamentar ninguna catástrofe, pero hubo miles de predicciones apocalípticas sobre las consecuencias que se podría llegar a obtener este error, así podría haber sido si no se hubiera reparado a tiempo.  Es por eso, el software tiene un papel muy importante en la sociedad sobre manera garantizar [métodos](https://www.ecured.cu/M%C3%A9todos) trasparentes en sus diferentes fases de producción y explotación.  **MODELOS DEL PROCESO DE DESARROLLO SOFTWARE**  No existe consenso sobre cuál es el mejor modelo del proceso software. Distintos equipos de desarrollo pueden utilizar diferentes modelos de proceso software para producir el mismo tipo de sistema [software](https://www.ecured.cu/Arquitectura_de_software). Sin embargo, algunos modelos son más apropiados para producir ciertos tipos de sistemas, de forma que si no se utiliza un modelo adecuado puede ocurrir que el sistema software resultante sea de menor calidad.  El reparto de costes entre las distintas fases del proceso de desarrollo es difícil de determinar dado los distintos modelos de proceso existentes. Sin embargo, en dependencia del modelo que se adopte, al menos el 60% del coste total se emplea en la actividad de evolución del sistema. La estimación de este porcentaje es pesimista, ya que la tasa de crecimiento de nuevos productos software es mucho mayor que la tasa de productos software que quedan en desuso (no tienen que ser mantenidos), por lo que el número de operaciones de mantenimiento que se realizan sigue aumentando. El proceso de diseño software debería, por tanto, tener en cuenta la posterior evolución del sistema.  Las características deseables de un proceso de desarrollo software son: Claridad: El proceso de desarrollo es claro cuando se entiende con facilidad. Visibilidad: Un proceso de desarrollo es visible cuando sus actividades producen resultados claros identificables externamente.  Facilidad de soporte: Exige disponer de herramientas CASE (Computer-Aided Software Engineering) que den soporte a todas o alguna de las actividades del proceso de desarrollo.  Fiabilidad: Un proceso de desarrollo es fiable cuando es capaz de detectar posibles errores.  **Facilidad de mantenimiento**: Requiere capacidad para incorporar nuevos requisitos o modificar alguno o algunos de los existentes.  Rapidez: Un proceso software es rápido cuando se puede obtener, a partir de la especificación, una implementación del sistema en un tiempo reducido.  **Modelo en cascada o convencional**  Tomado de otras ingenierías es el primer modelo de desarrollo software propuesto. Ampliamente usado en la industria por su facilidad de gestión y visibilidad. En la figura 1 se representa el secuenciamiento de las actividades de este modelo de desarrollo.  **MODELO EVOLUTIVO**  En este modelo se entrelazan las actividades de especificación, desarrollo y validación. Inicialmente, se desarrolla rápidamente un sistema inicial a partir de una especificación muy abstracta. El sistema se va refinando con la información que van suministrando los clientes y/o usuarios hasta que se obtiene un sistema final que satisfaga todas las necesidades previstas. El sistema final obtenido puede rediseñarse para producir otro más robusto y más fácil de mantener. En la figura 2 se esquematiza este modelo  <https://www.ecured.cu/images/7/71/Pds1.JPG>  Existen dos tipos de procesos de desarrollo evolutivos: Exploratorio: Su objetivo es trabajar con el cliente para identificar y construir el sistema final a partir de una especificación informal. El resultado del proceso es el sistema final.  Prototipado desechable: Su objetivo es entender los requisitos del cliente. El resultado del proceso es la especificación del sistema (el prototipo se deshecha). Los principales problemas de este modelo son: escasa visibilidad; los continuos cambios que hacen que los sistemas desarrollados estén deficientemente estructurados; y la necesidad de disponer, en muchos casos, de un equipo de desarrollo altamente calificado. Estos problemas hacen que la aplicación de este modelo se suela limitar a sistemas interactivos de tamaño pequeño o mediano. La deficiente estructura dificulta las tareas de mantenimiento de ahí que se suela aplicar a sistemas con una vida corta y a partes de grandes sistemas, especialmente a sistemas de inteligencia artificial y a interfaces de usuario.  **MODELO TRANSFORMACIONAL.**  Se basa en disponer de una especificación formal del sistema y en transformar, con métodos matemáticos, esta especificación en una implementación. Si las transformaciones que se aplican son correctas es posible asegurar que el sistema construido satisface la especificación, es decir, es posible obtener programas correctos por construcción.  **MODELO EN ESPIRAL**  Desarrollado por Boehm en el año 1988 con el objetivo de reunir las ventajas de los modelos de proceso software en cascada y de prototipado. Se incluye el análisis de riesgo como una parte importante del proceso de desarrollo software. El modelo tiene la forma de una espiral en la que cada vuelta representa cada una de las fases en las que se estructura el proceso software y está organizada en cuatro sectores:  1. Definición de objetivos, alternativas y restricciones de cada fase del proyecto. 2. Evaluación de alternativas y análisis de riesgos.  3. Desarrollo y validación. Se elige el modelo de proceso de desarrollo que se considere más adecuado.  4. Planificación de las siguientes fases del proyecto.  Recuperado de : <https://www.ecured.cu/Desarrollo_de_software> |  |

|  |  |
| --- | --- |
| Sección de texto (Sección Multimedia) | Sección de imágenes |
|  |  |

|  |  |
| --- | --- |
| Sección de texto | Sección de imágenes |
| **Referencias consultadas**  Cáceres, M. L. (2007). *La importancia de la Ingeniería de Software en el.* Mexico: Universidad Veracruzana (UV). Av. Xalapa esq. Av. Ávila Camacho s/n Col. Obrero .  Ecuador, P. (03 de Abril de 2013). *YouTobe.* Obtenido de Introducción al Desarrollo ágil de software desde la perspectiva del PMBOK: https://youtu.be/f-WjMWu8-b4  virguez, l. r. (24 de Octubre de 2016). *YouTobe*. Obtenido de INTRODUCCIÓN AL DESARROLLO DEL SOFTWARE.: https://youtu.be/7Bzdo0KZoLo  Zuluaga, J. C. (09 de 2009). *You Tube.* Obtenido de Desarrollo de Software Parte 1.  Zuluaga, J. C. (22 de Agosto de 2013). *You Tube.* Obtenido de Desarrollo de Software Parte 1.1: https://youtu.be/DQeOdu-K-Dc  Zuluaga, J. C. (09 de Febrero de 2014). *You Tube.* Obtenido de Desarrollo de Software Parte 1: https://youtu.be/XQL2jrTZPxM |  |

[**Pasos del desarrollo de software.**](http://www.mailxmail.com/curso-aprende-programar/pasos-desarrollo-software)

1. Especificación del programa  
2. Diseño del programa  
3. Codificación del programa  
4. Prueba  
5. Documentación  
6. Mantenimiento

1. Especificación del programa   
Se conoce también como definición del problema o análisis del programa. En este paso se determinan la información inicial para la elaboración del programa. Es donde se determina qué es lo que debe resolverse con el computador, de qué presupuestos se debe partir... en definitiva, el planteamiento del problema.

Se requieren cinco tareas:

a. Determinación de objetivos del programa.

Debe definirse claramente los problemas particulares que deberán ser resueltos o las tareas que hay que realizar, esto nos permitirá saber qué es lo que se pretende solucionar y nos proporcionará información útil para el planeamiento de la solución.

b. Determinación de la salida deseada.

Los datos seleccionados deben ser arreglados en una forma ordenada para producir información. Esta salida podría ser una salida de impresión o de presentación en el monitor.

c. Determinación de los datos de entrada.

Una vez identificada la salida que se desea, se pueden determinar los datos de entrada y la fuente de estos datos. Los datos deben ser recolectados y analizados.

d. Determinación de los requerimientos de procesamiento.   
Aquí se definen las tareas de procesamiento que deben desempeñarse para que los datos de entrada se conviertan en una salida.

e. Documentación de las especificaciones del programa.

Es importante disponer de documentación permanente. Deben registrarse todos los datos necesarios para el procesamiento requerido. Esto conduce al siguiente paso del diseño del programa.

2 . Diseño del programa   
Es diseñar cualquier sistema nuevo o las aplicaciones que se requieren para satisfacer las necesidades. Esta actividad se debe dividir en:  
- Operaciones de entrada/salida  
- Cálculos  
- Lógica/ comparación  
- Almacenamiento/ consulta

En este paso se genera una solución con técnicas de programación como diseño descendente de programas, pseudocódigos, flujogramas y estructuras lógicas.

3. Codificación del programa   
Es la generación real del programa con un lenguaje de programación. En esta etapa se hace uso de la lógica que desarrolló en el paso del diseño del programa para efectivamente generar un programa. Se debe seleccionar el lenguaje apropiado para resolver el problema.

4. Prueba y depuración del programa   
Depurar es correr el programa en una computadora y corregir las partes que no funcionan. En esta fase se comprueba el funcionamiento de cada programa y esto se hace con datos reales o ficticios. Cuando los programas están depurados, se prueban. Cuando los programas se depuran, se pueden encontrar los siguientes errores:

a) Errores de sintaxis o de compilación  
b) Errores de ejecución  
c) Errores de lógica  
d) Errores de especificación.

a) Errores de sintaxis o de compilación   
Es una violación de las reglas del lenguaje de programación. Son más fáciles de corregir, ya que son detectados por el compilador (posible error de escritura), el cual dará información sobre el lugar donde está y la naturaleza de cada uno de ellos mediante un mensaje de error.

b) Errores de Ejecución   
Se deben generalmente a operaciones no permitidas como dividir por cero, leer un dato no numérico en una variable numérica, exceder un rango de valores permitidos, etc. Se detectan porque se produce una parada anormal del programa durante su ejecución.

c) Errores de Lógica   
Corresponden a la obtención de resultados que no son correctos y la única manera de detectarlos es realizando suficientes pruebas del programa. Son los más difíciles de corregir, no sólo por la dificultad de detectarlos, sino porque se deben a la propia concepción y diseño del programa.

d) Errores de Especificación   
Es el peor tipo de error y el más difícil de corregir. Se deben a mal diseño del programa posiblemente por mala comunicación usuario programador y se detectan cuando ya se ha concluido el diseño e instalación del programa, lo cual puede implicar repetir gran parte del trabajo realizado.

Prueba : :   
Consiste en verificar la funcionalidad del programa a través de varios métodos para detectar errores posibles.  
Métodos de Prueba:  
Chequeo de escritorio  
Prueba manual de datos de muestra  
Intento de traducción  
Prueba de datos de muestra en la computadora  
Prueba por un grupo selecto de usuarios potenciales.

a. Chequeo de Escritorio:  
El programador se sienta frente a un escritorio y corrige una impresión del programa. Revisa el listado línea por línea en busca de errores de sintaxis y lógica.

b. Prueba manual de datos de muestra:  
Se corre el programa en forma manual aplicando datos tanto correctos como incorrectos para comprobar que funciona correctamente.

c. Intento de Traducción:  
El programa corre en una computadora usando un programa traductor para convertirlo a lenguaje de máquina. Para ello debe estar ya libre de errores de sintaxis, de lo contrario serán identificados por el programa de traducción.

d. Prueba de datos de muestra en la computadora:  
Después del intento de traducción y corregidos los errores de sintaxis, se procede a buscar errores de lógica utilizando diferentes datos de muestra.

e. Prueba por un grupo selecto de usuarios potenciales:  
Esto se conoce como prueba beta. Se trata por lo general del paso final en la prueba de un programa. Usuarios potenciales ponen a prueba el programa y ofrecen retroalimentación.

5. Documentación del programa   
Consiste en describir por escrito a nivel técnico los procedimientos relacionados con el programa y su modo de uso. También se debe documentar el programa para que sea más entendible.

¿Para quiénes son la documentación?   
- Usuarios (Digitadores)  
- Operadores  
- Programadores  
- Analistas de sistemas

Documentos que se elaboran:  
Manual de Usuario y Manual del Analista.  
A los usuarios se les elabora un manual de referencia para que aprendan a utilizar el programa. Esto se hace a través de capacitaciones y revisión de la documentación del manual de usuario. El manual del usuario no está escrito a nivel técnico sino al de los distintos usuarios previstos y explica en detalle cómo usar el programa: descripción de las tareas que realiza el programa, instrucciones necesarias para su instalación puesta en marcha y funcionamiento, recomendaciones de uso, menús de opciones, método de entrada y salida de datos, mensajes de error, recuperación de errores, etc.

A los operadores por si se presentan mensajes de error, sepan cómo responder a ellos. Además que se encargan de darle soporte técnico al programa.

A los programadores a través del manual del analista para que recuerden aspectos de la elaboración del programa o en caso que otras personas puedan actualizarlo o modificarlo (darle mantenimiento) y no son necesariamente las personas que lo diseñaron. Es por ello, que la documentación debe contener algoritmos y flujogramas de los diferentes módulos que lo constituyen y las relaciones que se establecen entre ellos; listados del programa, corridas, descripción de variables que se emplean en cada módulo, cuáles son comunes a diferentes módulos y cuáles locales; descripción de los ficheros de cada módulo y todo lo que sea de importancia para un programador.

A los analistas de sistemas que son las personas que deberán proporcionar toda la información al programador. Estos se encargan de hacer una investigación previa de cómo realizar el programa y documentar con las herramientas necesarias para que el programador pueda desarrollar el sistema en algún lenguaje de programación adecuado.

6. Mantenimiento del programa   
Es el paso final del desarrollo del software. Alrededor del 75% del costo total del ciclo de vida de un programa se destina al mantenimiento. El propósito del mantenimiento es garantizar que los programas en uso estén libres de errores de operación y sean eficientes y efectivos.