Universidad Nacional de La Matanza

Departamento Ingeniería e Investigaciones Tecnológicas

Jefe de Cátedra
Mg. Gabriela GABAY / Ing. Gonzalo PARADELA

Requerimientos para la Ingeniería

Apunte bibliográfico

Código de Materia 1108

Edición
03/2021

Profesores

Mg. Gabriela Gabay
Lic. Carla Crocco
Lic. María Laura Pepe
Ing. Gonzalo Paradela

UNIDAD 3

PROCESO SOFTWARE

Proceso de resolución de problemas

El proceso de resolución de problemas sigue las siguientes etapas:

- Identificar el problema
- Definir cómo resolver el problema
- Resolver el problema
- Probar el resultado
- Usar el resultado

La secuencia anterior describe el proceso seguido por la mente, y esto ciertamente ayuda, pero son necesarios muchos conocimientos del dominio o área concreta donde se plantea el problema, para poder aplicar este proceso de modo que lleve a una solución.

El proceso de la construcción de software

La construcción de un software es también una actividad de resolución de problemas. Tiene dos objetivos. Por un lado, dada una necesidad, pretende satisfacerla mediante una solución tratable por computadora.

Por otro lado, el subsecuente mantenimiento del software producido hasta el final de su vida útil. Entonces podemos decir que el proceso de software es la transformación de una necesidad (problema) en un software que satisface esa necesidad.

La solución de un problema mediante ingeniería de software es una actividad de modelización que comienza con el desarrollo de modelos conceptuales (no formales) y los convierte en modelos formales, que son los productos implementados. El modelo conceptual se corresponde con el punto de vista que las personas tienen del problema. El modelo formal concierne a la perspectiva que de ese mismo problema tiene la computadora.

Debe existir entonces un proceso de resolución que se corresponda con el proceso de resolución básico de problemas. En efecto, al primer paso o “identificar el problema” se le denomina análisis y especificación de requisitos. Luego diseño del sistema de software. El tercer paso es la codificación. Posteriormente el sistema es sometido a pruebas y finalmente el software debe ser instalado.

Al proceso de resolución de problemas de software debe añadírsele la etapa de mantenimiento.

Análisis y especificación de requisitos

Incluye el análisis del problema y concluye con una especificación completa del comportamiento externo que debería tener el sistema a construir.

Diseño del sistema

El diseño se realiza a alto nivel o diseño preliminar (incluye los componentes principales) y a bajo nivel o diseño detallado (se definen y documentan los algoritmos que llevarán a cabo las funciones de cada componente o módulo).

Codificación

Consiste en transformar el diseño detallado en un lenguaje comprensible por la computadora. Se lleva a cabo en dos niveles, la conversión del algoritmo en lenguaje de alto nivel, y la transformación del lenguaje de alto nivel a lenguaje máquina. La primera transformación la realizan personas/desarrolladores y la segunda transformación la realiza un compilador.

Pruebas

Si los humanos fueran perfectos, el proceso podría terminar en el punto anterior. Como no es el caso, se realizan pruebas a distintos niveles.

- Pruebas unitarias: se prueba cada módulo por separado verificando que realiza lo que pedía el diseño de bajo nivel
- Pruebas de integración: comprueba el funcionamiento de un conjunto de módulos probados en las pruebas unitarias. Lo ideal es que cada conjunto de módulos se corresponda con un componente del diseño de alto nivel.
- Pruebas de sistema: aseguran que la totalidad del sistema software se comporte de acuerdo con la especificación de requisitos inicial.
- Pruebas de regresión: en sistemas existentes, estas pruebas, aseguran que los componentes que funcionaban antes de una evolución siguen funcionando.

Instalación

Tras las pruebas, el sistema software y su entorno hardware pasan a la fase operativa.

Mantenimiento y ampliación

Consiste en la detección continuada de errores y su reparación. La ampliación consiste en la adición al sistema de nuevas capacidades.

Lo único constante es el cambio

Durante la última década, la informática en su conjunto se ha ido simplificando, sin embargo, los entornos de desarrollo han seguido un camino contrario y son cada día más complejos.

Hace algún tiempo, un desarrollo típico estaba formado por un par de docenas de transacciones, algunos utilitarios y poco más. Todo era sencillo porque había pocas cosas donde elegir, casi siempre la elección dependía del propio suministrador del hardware y tales plataformas de desarrollo eran estables y no se producían cambios durante algún tiempo importante.

El trabajo era realizado de manera *"artesanal"* por un programador, que interactuaba directamente con el cliente, y obtenía por el mismo la retroalimentación para medir la calidad.

Hoy en día todo esto ha cambiado; las plataformas de desarrollo proliferan y ofrecen nuevas posibilidades que hacen que la oferta sea más atractiva y se cambia de entornos en función de las nuevas posibilidades que los mismos ofrecen.

Los usuarios son mucho más exigentes, demandan unas prestaciones antiguamente no soñadas, desde nuevos dispositivos móviles, y esto hace que en muchas ocasiones el diseño gráfico y su interfaz genere más trabajo de programación que los propios algoritmos para los que se concibe el programa.

La necesidad de desarrollar un software normalmente multiplataforma dificulta, no sólo el propio desarrollo sino también las pruebas de aceptación de este. Hoy no basta con que una aplicación funcione, sino que debe funcionar en diferentes sistemas operativos y bajo diferentes condiciones.

Como si no tuviéramos demasiado con tener que solucionar los problemas de hoy, aparecen las preguntas sobre la tecnología del mañana: ¿Qué sistemas tendremos dentro de unos años?, ¿cómo conseguir un entorno de desarrollo del que se pueda migrar fácilmente al acontecer futuro?, ¿Qué haremos con las aplicaciones existentes en un futuro?, ¿Serán obsoletas?

Estas preguntas que nos formulamos ahora mismo deberán ser consideradas por el director del proyecto para conocer el grado de realización del software con vistas a un mantenimiento futuro.

Evolución histórica del concepto de calidad en la industria del software

La industria del software es una industria joven que ha evolucionado rápidamente, aunque no tanto como para alcanzar la madurez que tiene la industria tradicional.

Es importante recordar que es evidente la influencia de la industria tradicional sobre la industria del software. La siguiente tabla propuesta por John J. Marciniak, pone de manifiesto esta relación, comparando las diferentes fases superadas por el control de calidad:

Etapa	Descripción	Industria del Software	Industria en general
Artesanos	Se fían de la creatividad y del buen trabajo artesanal	Años ‘60	Antes del siglo XIX
Inspección	Supervisores inspeccionan la calidad antes de la liberación del producto	Años ‘70	Siglo XIX
Control estadístico del proceso	Cuantificación de la calidad del producto, técnicas de muestreo	Pocas evidencias de uso	Años ‘30
Aseguramiento de la calidad	Uso de estándares en los sistemas de calidad para los procesos	Años ‘80	Años ‘50
Conformidad con la calidad	Calidad total: se eliminan derroches y minimizan costes	Años ‘90	Años ‘80
Calidad dirigida al cliente	Calidad total dirigida hacia el cuidado del cliente y del servicio	Pocas evidencias de uso	Años ‘90
Calidad dirigida al mercado	Calidad total dirigida hacia el cliente existente, así como a clientes en potencia	Pocas evidencias de uso	Algunas evidencias de uso

CALIDAD DEL SOFTWARE

La Calidad del Software es una preocupación a la que se dedican muchos esfuerzos, pero aun así se acepta que el software casi nunca es perfecto. De acuerdo con lo anteriormente expuesto la calidad del servicio o producto orientada a satisfacer las necesidades del cliente es el aspecto sobre el cual se centra la industria del software. Ahora bien, dado que el software es un producto virtual que no se fabrica, ni que tampoco se degrada físicamente, sino que se desarrolla en base a interacciones humanas (ej. Elicitación de requerimientos) debe seguir normas particulares que conduzcan al desarrollo de lo que comúnmente se conoce como un Software de Calidad.

Ya sea un producto o servicio, el software no permite una medición tradicional de calidad, debido a la naturaleza anteriormente expuesta. Por esta razón la certificación de la calidad se da sobre los procesos y la correcta ejecución y reiteración de estos.

Dado que entonces el poder de medición de la calidad del software radica en el usuario, en ese sentido la calidad depende de quien la juzgue. El hecho de que una empresa tenga certificación en calidad de software no garantiza que su software sea de calidad.

CMMI

CMMI Es la sigla del modelo utilizado en la Industria del Software para evaluar si una empresa mantiene cierto nivel de calidad en relación con el software. CMMI (Capability Maturity Model Integration) es un estándar que cuenta con muchas empresas seguidoras, igualmente es importante mencionar que existen otras normas o modelos tales como: ISO 90001, ITIL, SPICE, etc.

Cuando una organización quiere acreditarse como cumplidora del modelo CMMI debe pasar por una evaluación la cual especifica en cuál de los diferentes niveles dentro del modelo se encuentra. Según el nivel en que se encuentre una empresa, tendrá que cumplir con requerimientos más o menos exigentes. Su implementación no se orienta solo a grandes empresas, ya que un emprendimiento con 5 empleados es suficiente para certificar CMMI.

El modelo CMMI es gratuito y se puede descargar desde la web. Su implementación se realiza a través de la utilización de métricas para comprobar que se producen cambios reales en el software que produce la empresa. Si implantar un sistema de este tipo no deriva en mejoras que se puedan verificar, muchas empresas no lo adoptarían. Algunas métricas o indicadores pueden ser:

1. Índice de productividad = tamaño / esfuerzo = líneas de código fuente / horas trabajadas
2. Tasa de defectos = defectos / tamaño = número de errores / líneas de código generadas

Tomar el timón del barco

Podríamos comparar un proyecto software con un barco que está zarpando. Si no se han planificado bien los recursos previamente, será muy difícil que se pueda llegar a destino. Se deben detectar y organizar todas las tareas. A cada tarea se le debe asignar recursos materiales y humanos para que pueda ser ejecutada en un determinado periodo, considerando siempre un uso eficiente de los recursos. En el caso de que la planificación haya sido correcta y el proyecto comenzó (el barco ha zarpado), será muy importante un control y seguimiento continuo de los recursos humanos y materiales en el transcurso del tiempo. Una mala administración de estos podrá hacer que los recursos se acaben antes de llegar a destino. Esto dejará al barco (proyecto) varado en medio del océano.

Existen diferentes técnicas para hacer el seguimiento de un proyecto y por lo tanto es importante que el capitán del barco, es decir el director del proyecto, las conozca y las aplique. Generalmente estas técnicas son útiles y necesarias, casi tanto como una brújula en el mar. En el mejor caso en que la planificación, el seguimiento y el control sean ejecutados efectivamente, nada nos libra de que una noche cualquiera nos llegue una tormenta que pueda desestabilizar el viaje y el barco se hunda, es decir que el proyecto pueda fracasar.

Estas podrían ser todas las variables externas que podrían afectar el proyecto: como un aumento inflacionario, una huelga general, aumento excesivo en los precios de producto, etc. En este caso se deberá tener un buen análisis de riesgo para actuar ante cualquier evento con un plan de contingencia apropiado. Es importante que un director de proyectos informáticos incorporé herramientas gerenciales para poder planificar y ejecutar el proyecto de una manera profesional.

Proyecto

Un proyecto es un esfuerzo temporal acometido para crear un único servicio o producto.

“Conjunto de actividades planificadas, ejecutadas y supervisadas con el fin de alcanzar un fin común con recursos finitos”

Hay que recordar que uno de los recursos finitos más importante es el tiempo. La naturaleza temporal de los proyectos indica un principio y un final definidos. El final se alcanza cuando se logran los objetivos del proyecto o cuando se termina el proyecto porque sus objetivos no se cumplirán o no pueden ser cumplidos, o cuando ya no existe la necesidad que dio origen al proyecto.

Tipos de Proyectos Informáticos

Existen diferentes clasificaciones de los tipos de proyectos informáticos. A continuación, listamos los principales tipos de proyectos informáticos:

- Software
  - Metodologías, Ingeniería del software, etc.
  - Software empotrado.
- Hardware
  - Velocidad de Proceso, S.O., Servicios, etc.
- Comunicaciones y Redes
  - Protocolos, Buses, Cableado, etc.
- Instalaciones de Hardware
  - Peso de los equipos, Instalación de aire acondicionado, Suelo flotante, Extinción de incendios
- Sistemas de Misión Crítica
  - Industrial, Médica, Nuclear, Militar, Aeronáutica, etc.
  - Tiempo real, Esquemas productivos, etc.
- Auditorías
  - Sistemas, Seguridad, Calidad, Legislación …
- Peritajes
  - Civiles, Penales, Laborales…

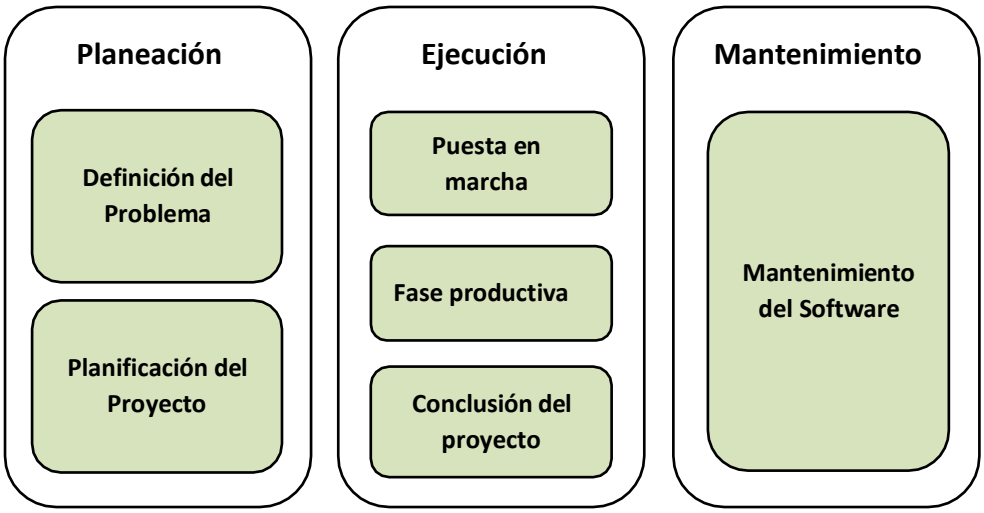
- Consultoría y Asesoría
  - Sobre cualquier actividad.
- Seguridad Informática (ISO 17799)
  - Seguridad de la Información.
- Reingeniería de Proyectos
  - De cualquiera de los tipos

Diferencia de proyecto informático con proyectos tradicionales

En los proyectos de ingeniería, la buena calidad se adquiere mediante un buen diseño, pero en el caso del software la etapa de construcción incide pobremente en su calidad, no así en la construcción de hardware o una obra civil. Otra diferencia es que el software no se estropea, el paso del tiempo o males del entorno no inciden en el aumento de tasa de fallos. Entonces, no se puede gestionar un proyecto de software como si fuese un proyecto de fabricación.

Fases de un proyecto

En la mayoría de los proyectos se pueden distinguir tres grandes secciones de trabajo:



Planeación

En esta etapa es necesario tener claro tanto el problema que se pretende solucionar, el producto que se quiere obtener, como el servicio que se quiere proporcionar. A su vez, es necesario evaluar tanto los costos económicos como los recursos humanos.

Fases de la Planeación

- *Inicio y definición del problema*: clarificar el servicio o producto a obtener. En este punto hay que trabajar con los usuarios, directores y clientes, ellos son quienes nos dirán dónde está el problema. Hay que tener en cuenta que todo el proyecto se basará en esta definición, y es mejor que quede clara. En esta fase también se definen los límites del proyecto. En esta etapa se deben realizar las siguientes tareas:
  - Estimar el tamaño de la aplicación
  - Identificar las tareas a realizar
  - Asignar recursos a cada tarea
  - Crear un calendario de tareas
  - Realizar un estudio económico

- *Planificación del Proyecto*: esta etapa atiende las necesidades que aparecerán a lo largo del desarrollo, anticipando el curso de las tareas a realizar, la secuencia en que se llevarán a cabo, los recursos y el momento en que serán necesarios. Para poder identificar todas las cosas necesarias para poder alcanzar el objetivo marcado, se consideran tres dimensiones: Calidad, Costo, Duración.

Ejecución

En esta fase se trata de llevar a cabo el plan previo. La ejecución es fuertemente influida por la planificación, es decir, una mala planificación supondrá una mala ejecución.

Fases de la Ejecución

- *Puesta en marcha*: en esta fase se debe organizar el equipo de desarrollo, los mecanismos de comunicación, la asignación de roles y de responsabilidades de cada persona. Las principales tareas son:
    - Ajustar las etapas planificadas de acuerdo con la disponibilidad de personal
    - Establecer una estructura organizativa
    - Definir responsabilidades y autoridad
    - Organizar el lugar de trabajo
    - Puesta en funcionamiento del equipo
    - Divulgación de los estándares de trabajo y sistemas de informes

- *Fase productiva*: aquí se realiza la actividad gruesa del proyecto. Se realiza análisis, desarrollo (ver paradigmas) diseño, implementación y pruebas. En este momento, no deberían existir dudas sobre las especificaciones, recursos, y tareas. El responsable del proyecto debe realizar tareas de seguimiento para poder identificar cualquier tipo de dificultad y poder gestionar los riesgos que pudieran aparecer. De aparecer alguna situación, el jefe de proyecto deberá realizar acciones correctivas o preventivas para corregir el curso del proyecto.

- *Conclusión del proyecto*: se da por terminado el proyecto y se entrega el producto/servicio terminado al cliente. Las principales actividades son:
    - Revisar las desviaciones e identificarlas para evitarlas en futuros proyectos
    - Reasignar el personal a nuevos proyectos o devolverlos a su área
    - Documentar las relaciones entre los empleados para futuros proyectos

Mantenimiento

Es el proceso de mejora y optimización del software después de su entrega al usuario final, así como también la corrección y prevención de defectos.

Ciclos de vida del proceso de Software

Existen varios modelos de ciclo de vida, entre los que se destacan: cascada, prototipado de usar y tirar, incremental, emisión gradual, mejora iterativa, ensamblaje de componentes, espiral, prototipado operativo y prototipado rápido. No existe un modelo de ciclo de vida que funcione para cualquier proyecto.

El Ingeniero en software debe estudiar las características del proyecto y seleccionar el modelo de ciclo de vida que más se adapte a ellas. Las bases para determinar el ciclo de vida más adecuado son: la cultura de la organización, la disponibilidad para correr riesgos, el dominio de la aplicación, la volatilidad de los requisitos y cuánto se comprenden dichos requisitos.

Esta selección se realiza al inicio del proyecto y se deben considerar un conjunto de variables de análisis que requiere en su mayor parte de la experiencia del ingeniero en software, ya que al inicio del proyecto subyace una gran cuota de subjetividad en casi todos los aspectos del proyecto.

El proceso de construcción de software puede verse como una cadena de tareas. Las cadenas de tareas son planes idealizados de qué acciones deben realizarse y en qué orden. El software obtenido tras el proceso puede ser visto como el "producto" que entra al proceso, se transforma (a lo largo de la cadena de tareas) y que sale del proceso hasta obtener el producto deseado. Desde esta perspectiva del producto, se pueden establecer los estados por los que va pasando el producto en un proceso software: la entrada al proceso es una necesidad, que una vez estudiada se convierte en una especificación de requisitos, que posteriormente se transforma en un diseño del sistema, para pasar más adelante a ser un código y finalmente un sistema software completo e integrado. Este enfoque orientado al producto, focalizado en el producto transformado (en lugar del proceso que lo transforma) se llama ciclo de vida. Es decir, el ciclo que el producto software sufre a lo largo de su vida, desde que nace (o se detecta la necesidad) hasta que muere (o se retira el sistema).

Paradigmas de programación

Un paradigma de programación provee (y determina) la visión y métodos de un programador en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas (con la solución de múltiples "problemas" se construye una aplicación).

Los lenguajes de programación son basados en uno o más paradigmas, por ejemplo: Smalltalk y Java son lenguajes basados en el paradigma orientado a objetos. El lenguaje de programación Scheme, en cambio, soporta sólo programación funcional. En cambio, Python, soporta múltiples paradigmas.

Clasificación por paradigmas de programación

- *Paradigma Estructurado:* la programación se divide en bloques (procedimientos y funciones) que pueden o no comunicarse entre sí. Además, la programación se controla con secuencia, selección e iteración. Permite reutilizar código programado y otorga una mejor compresión de la programación. Es contrario al paradigma "inestructurado", de poco uso, que no tiene ninguna estructura, es simplemente un "bloque", como, por ejemplo, los archivos batch (.bat).

- *Paradigma Orientado a Objetos*: está basado en la idea de encapsular estado y operaciones en objetos. En general, la programación se resuelve comunicando dichos objetos a través de mensajes (programación orientada a mensajes). Se puede incluir -aunque no formalmente- dentro de este paradigma, el paradigma basado en objetos, que además posee herencia y subtipos entre objetos. Ej.: Simula, Smalltalk, C++, Java, Visual Basic .NET, etc. Su principal ventaja es la reutilización de códigos y su facilidad para pensar soluciones a determinados problemas.

- *Paradigma Funcional*: este paradigma concibe a la computación como la evaluación de funciones matemáticas y evita declarar y cambiar datos. En otras palabras, hace hincapié en la aplicación de las funciones y composición entre ellas, más que en los cambios de estados y la ejecución secuencial de comandos. Permite resolver ciertos problemas de forma elegante y los lenguajes puramente funcionales evitan los efectos secundarios comunes en otro tipo de programaciones.

- *Paradigma lógico*: se basa en la definición de reglas lógicas para luego, a través de un motor de inferencias lógicas, responder preguntas planteadas al sistema y así resolver los problemas. Ej.: prolog (todo humano es mortal, Sócrates es humano, entonces Sócrates es mortal, eso sería, humano(x) => mortal(x), humano (Sócrates) entonces mortal (Sócrates)).

Otros paradigmas y sub-paradigmas son: paradigma imperativo, declarativo, orientado al sujeto, paradigma reflectante, programación basada en reglas, paradigma basado en restricciones, programación basada en prototipos, paradigma orientado a aspectos, etc.