UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA

ESCUELA DE CIENCIAS BASICAS, TECNOLOGÍA E INGENIERIA

PROGRAMA INGENIERIA DE SISTEMAS

MODULO

Ingeniería de Software

Autor: Ing. Alexandra Aparicio
Revisado y Editado: Ing. Jairo Martínez

Última Actualización:

Ing. Pilar Alexandra Moreno
Diciembre 2012

Tabla de Contenido

		Pág.
INTRODUCCIÓN		4
PRIMERA UNIDAD. INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE		6
INTRODUCCIÓN		6
1.	EL PRODUCTO	7
	1.1 El producto	7
	1.2 Evolución del Software	8
	1.3 El software	9
	1.4 Aplicaciones del Software	9
	1.5 Mitos del Software	11
2.	EL PROCESO	13
	2.1 Definición de Ingeniería del software	13
	2.2 Esquema de la Ingeniería del Software	15
	2.3 Esencia de la Ingeniería del Software	16
	2.4 Procesos, métodos y herramientas	17
	2.5 El proceso del software	19
3.	MODELOS DE PROCESO DE SOFTWARE	20
	3.1 Modelo lineal secuencial	20
	3.2 Modelo de construcción de prototipos	22
	3.3 Modelo DRA	25
	3.4 Modelos de procesos evolutivos de software	27
	3.5 Modelo de métodos formales y Técnicas de cuarta generación	31
SEGUNDA UNIDAD. GESTIÓN Y PLANIFICACIÓN DE PROYECTOS SOFTWARE		34
INTRODUCCIÓN		34
1.	CONCEPTOS SOBRE GESTIÓN DE PROYECTOS	35
	1.1 Gestión de proyectos	35
	1.2 Personal	36
	1.3 El producto	38
	1.4 El proceso	39
	1.5 El proyecto	40
2.	EL PROCESO DE SOFTWARE Y MÉTRICAS DEL PROYECTO	42
	2.1 Métricas en el proceso y dominios del proyecto	43
	2.2 Mejora estadística del proceso del software	45
	2.3 Métricas del proyecto	48
	2.4 Mediciones del software	49
	2.5 Métricas para la calidad del software	53
3.	PLANIFICACION DE PROYECTOS DE SOFTWARE	58
		Pág.
	3.1 Ámbito del software y Recursos	59

	3.2 Estimación del proyecto de software y Técnicas de Descomposición	61
	3.3 Modelos empíricos de Estimación	64
	3.4 Riesgo del Software	70
	3.5 Planificación temporal del proyecto	80
TERCERA UNIDAD. CONTROL DE CALIDAD DEL SOFTWARE		89
INTRODUCCIÓN		89
1.	GARANTIA DE CALIDAD DEL SOFTWARE	90
	1.1 Conceptos de calidad	91
	1.2 Tendencia de la calidad	94
	1.3 Garantía y aseguramiento de la calidad del software	96
	1.4 Revisiones del software	97
	1.5 Garantía de calidad estadística, Fiabilidad y Estándar ISO 9001	101
2.	TECNICAS DE PRUEBA DEL SOFTWARE	109
	2.1 Fundamentos de la prueba del software	109
	2.2 Diseño de casos de prueba, pruebas de la caja blanca y del camino básico	111
	2.3 Prueba de la estructura de control	114
	2.4 Prueba de caja negra	115
	2.5 Prueba de entornos especializados, arquitecturas y aplicaciones	116
3.	ESTRATEGIAS DE PRUEBA DEL SOFTWARE	119
	3.1 Enfoque estratégico la prueba del software	119
	3.2 Prueba de unidad	124
	3.3 Pruebas de integración del sistema	126
	3.4 Métricas técnicas del software	132
	3.5 Métricas del modelo del software	139
	RECURSOS DE SOFTWARE LIBRE PARA INGENIERÍA DEL SOFTWARE	147
	RECURSOS BIBLIOTECA VIRTUAL UNAD	154
	BIBLIOGRAFÍA	157

INTRODUCCIÓN

El curso Ingeniería de Software tiene como objetivo desarrollar habilidades y adquirir capacidades en la utilización de métodos y técnicas para desarrollar y mantener software de calidad.

El curso tiene 3 créditos académicos los cuales comprenden el estudio independiente y el acompañamiento tutorial, con el propósito de:

- Comprender los aspectos técnicos y de gestión de la disciplina de ingeniería de software.
- Capacitar a los estudiantes en las técnicas de gestión necesarias para planificar, organizar, supervisar y controlar proyectos de software.
- Fomentar en el estudiante técnicas de gestión de calidad del software.
- Obtener un conjunto de técnicas de prueba de software con el propósito de encontrar y corregir errores antes de entregar el software al cliente.

Este curso esta compuesto por tres unidades didácticas a saber:

Unidad 1. Introducción a la ingeniería de software: se presenta una vista general sobre la definición de: ingeniería de software, producto de software, procesos de software, se determina las características del software, los mitos del software. Se presenta también los diferentes tipos de proceso y los modelos evolutivos del software.

Unidad 2. Gestión y planificación de proyectos de software: se trata de determinar como se debe gestionar el personal, el proceso y el problema durante un proyecto de software. Se identifican las métricas de software y cómo pueden emplearse para gestionar el proceso de software y el proyecto llevado a cabo como parte del proceso.

Unidad 3. Control de calidad del software: se contemplan los aspectos relacionados con la calidad del software, se identifican los aspectos de gestión y las actividades específicas del proceso de calidad del software. Se establece la importancia de la garantía de calidad del software así como se definen las estrategias para los planes de garantía de calidad del software.

La ingeniería de software es el proceso de construir aplicaciones de tamaño o alcance prácticos, en las que predomina el esfuerzo del software y que satisfacen los requerimientos de funcionalidad y desempeño. La ingeniería de software, ofrece métodos y técnicas para desarrollar, mantener, producir y asegurar software de calidad.

Por tal razón, este curso teórico pretende describir los aspectos técnicos y de gestión de la Ingeniería de Software, así como de establecer la importancia de la garantía de calidad del software.

UNIDAD 1. INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE

INTRODUCCIÓN

La ingeniería de software es una disciplina que integra procesos, métodos y herramientas para el desarrollo de software. Varios son los modelos de procesos que se han propuesto para la ingeniería de software, cada uno presenta ventajas y desventajas, pero todos tienen en común fases genéricas que permiten llevar a cabo el proceso de la ingeniería de software.

OBJETIVOS

GENERAL

- Comprender los aspectos técnicos y de gestión de la disciplina de Ingeniería del Software

ESPECIFICOS

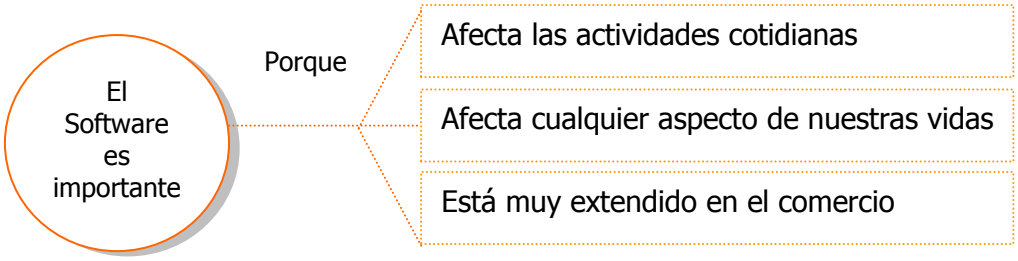
- Definición de Ingeniería de software, producto de software, procesos de software.
- Identificar los mitos de software
- Determinar que es un “proceso” de software
- Identificar los procesos que se pueden aplicar al desarrollo del software
- Determinar la diferencia entre modelos de proceso lineales e iterativos

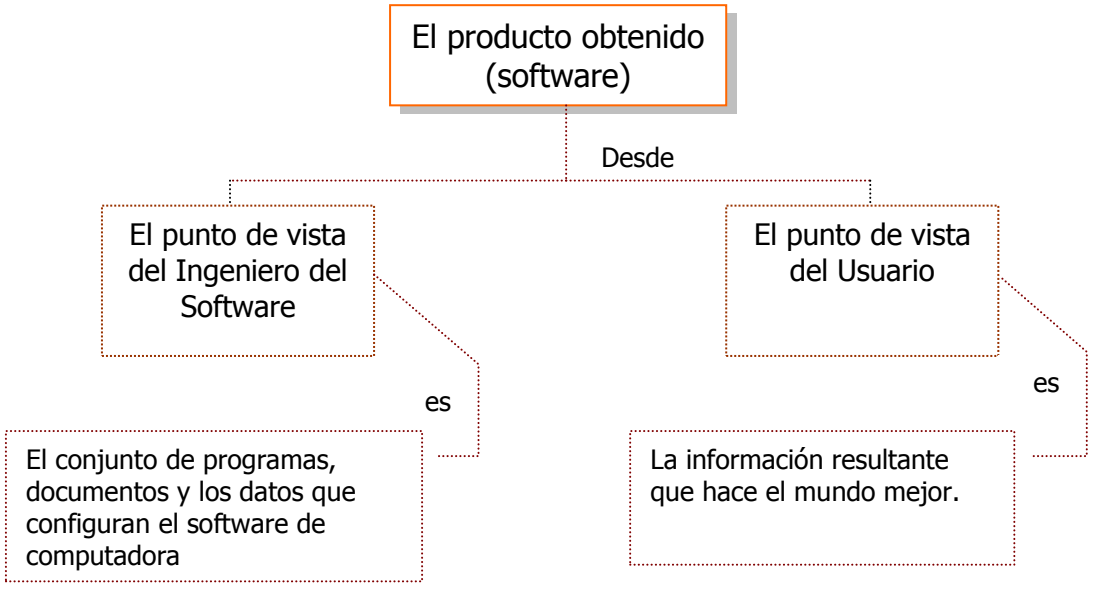
ESTRUCTURA TEMÁTICA

1. EL PRODUCTO

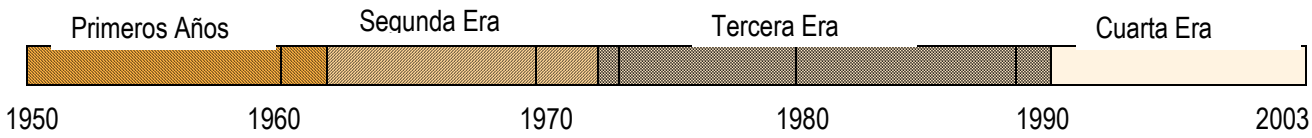
1.1 Definición del Producto Software.

El software es el producto que diseñan y construyen los ingenieros del software de cualquier tamaño y arquitectura.





1.2 La evolución del Software



[1]Primeros Años	Segunda era
• El software estaba en su infancia • El software era un añadido • Existían pocos métodos para la programación • No se tenia una planificación para el desarrollo del software • Los programadores trataban de hacer las cosas bien • El software se diseñaba a medida • El software era desarrollado y utilizado por la misma persona u organización (entorno personalizado) • El diseño de software era realizado en la mente de alguien y no existía documentación	• Aparece la multiprogramación y los sistemas multiusuario • Establecimiento del software como producto y la llegada de las casas de software • El software se desarrollaba para ser comercializado • Se empezó a distribuir software para grandes computadoras y minicomputadores • Comenzó a extenderse las bibliotecas de software • El mantenimiento de software comenzó a absorber recursos en una gran medida. • Comenzó una crisis del software porque la naturaleza personalizada de los programas hizo imposible su mantenimiento.
Tercera era	**Cuarta era**
• Complejidad alta en los sistemas informáticos • Sistemas distribuidos • Incorporación de “inteligencia” • Ejecución de funciones concurrentes • Desarrollo de software para redes y comunicaciones • Planificación en el proceso del desarrollo de software	• Impacto colectivo del software • Sistemas operativos operativos sofisticados , en redes globales y locales • Aplicaciones de software avanzadas • Entorno cliente/cliente servidor • Superautopista de información y una conexión del ciberespacio • La industria del software es la cuna de la economía • Tecnologías orientadas a objetos • Técnicas de cuarta generación para el desarrollo de software • Software de redes neuronales • Sistemas expertos e inteligencia artificial

[1] Roger S. Pressman. Ingeniería del software. Un enfoque práctico. Cuarta edición.

	• Programación de realidad virtual y sistemas multimedia • Algoritmos genéticos • Adopción de prácticas de Ingeniería del software

1.3 El Software

El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos, y por tal razón no se puede tomar como sólo el conjunto de programas, instrucciones y estructuras de datos. A continuación se presentan algunas características que permiten visualizar lo que en realidad es el software.

Características del Software

El software

- **Se desarrolla, no se fabrica:** se utiliza un modelo de proceso de desarrollo que comprende análisis, diseño, desarrollo, implementación y evaluación para obtener un producto de calidad.
- **No se "estropea", pero se deteriora:** El software durante su vida sufre cambios por lo que es probable que surjan fallos y defectos que si no se corrigen permiten que el software se vaya deteriorando.
- **Se construye a medida: a** medida que el software evoluciona se crean estándares de diseño. El software debe diseñarse e implementarse para que pueda ser reutilizable.

1.4 Aplicaciones del software

El software tiene una gran amplitud de aplicaciones. A continuación se relacionan:

Software de sistemas

Conjunto de programas creados como herramienta para otros programas. Por ejemplo: compiladores, Sistemas operativos

Software de gestión

Gestión de grandes cantidades de información almacenadas, para facilitar la toma de decisiones. Por ejemplo Bases de datos y aplicaciones de gestión de empresa

Software de ingeniería y científico

Utiliza algoritmos de manejo de números, simulación de sistemas, utiliza software en tiempo real. Por ejemplo: aplicaciones de astronomía, vulcanología, fabricación automática.

Software de tiempo real

El software que coordina / analiza/ controla sucesos del mundo real conforme ocurren, se denomina de tiempo real.

Software empotrado

Reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales. Por ejemplo, el control de las teclas de un horno de microondas, funciones digitales en un automóvil.

Software para PC

Aplicaciones orientadas a usuarios individuales o multiusuarios. Por ejemplo: procesadores de texto, hojas de cálculo, juegos, aplicaciones financieras, gestores de bases de datos.

Software de inteligencia artificial

Hace uso de algoritmos no numéricos para resolver problemas complejos. Por ejemplo: sistemas expertos, redes neuronales, robótica, prueba de teoremas y juegos.

1.5 Mitos del software

Los mitos de software propagaron información errónea y confusión, lo que conllevo a la crisis del software durante los primeros años del desarrollo del software.

Mitos de gestión

- Tenemos ya un libro que está lleno de estándares y procedimientos para construir software, ¿no le proporciona ya a mi gente todo lo que necesita saber?
- Mi gente dispone de las herramientas de desarrollo de software más avanzadas, después de todo, les compramos las computadoras más modernas.
- Si fallamos en la planificación, podemos añadir más programadores y adelantar el tiempo perdido.

Mitos del Cliente

- Una declaración general de los objetivos es suficiente para comenzar a escribir los programas.
- Los requisitos del proyecto cambian continuamente, pero los cambios pueden acomodarse fácilmente, ya que el software es flexible.

Mitos de los desarrolladores

- Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.
- Hasta que no tengo el programa "ejecutándose", realmente no tengo forma de comprobar su calidad.
- Lo único que se entrega al terminar el proyecto es el programa funcionando.

ACTIVIDADES COMPLEMENTARIAS

1. Muchos autores han tratado el impacto de la “era de la información”. Dé varios ejemplos (positivos y negativos) que indiquen el impacto del software en nuestra sociedad.

2. Escriba un informe que resuma las ventajas recientes en una de las áreas de aplicaciones de software principales. Entre las selecciones potenciales se incluyen: aplicaciones avanzadas basadas en Web, realidad virtual, redes neuronales artificiales, interfaces humanas avanzadas y agentes inteligentes.

3. Analice y describa la “Realidad” para cada uno de los mitos descritos en el numeral 1.5.

CONSULTAS WEB

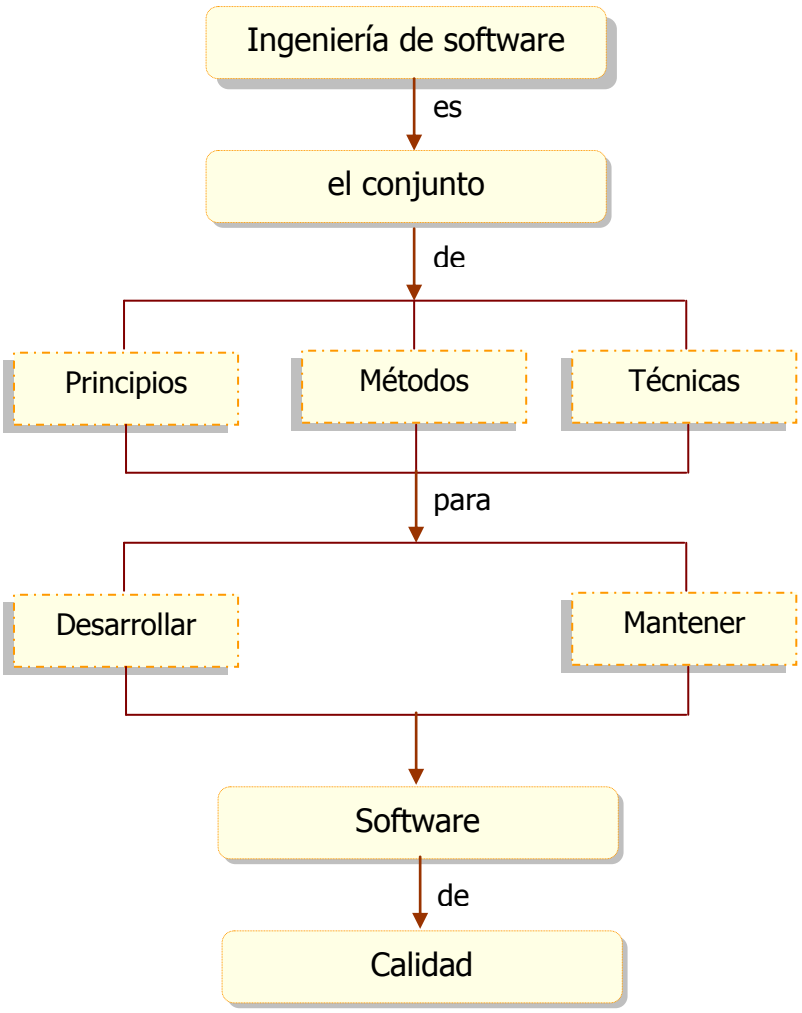
- http://www.rspa.com/spi/glossary.html , Glosario de términos de software.

- http://books.google.com.co/books?id=ytdKQGJ8f_AC&lpg=PA4&ots=hSqsOPw078&dq=Software%20Myths&pg=PA5 , encuentra un libro con información sobre los mitos del software.

2. EL PROCESO

Es una serie de pasos a seguir para construir un producto o un sistema. El proceso del software es importante porque proporciona estabilidad, control y organización a una actividad que puede, si no se controla, volverse caótica.

2.1 Ingeniería del Software



A nivel internacional, la Ingeniería de Software empieza a tomar un papel fundamental y como un área de la Ingeniería. A continuación se relacionan algunas definiciones de Ingeniería del Software:

1 **Zelkovitz. Principles of Software Engineering and Design.**
Ingeniería del software es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software.

2 **Boehm. Software Engineering.**
Ingeniería del software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar y mantenerlos.

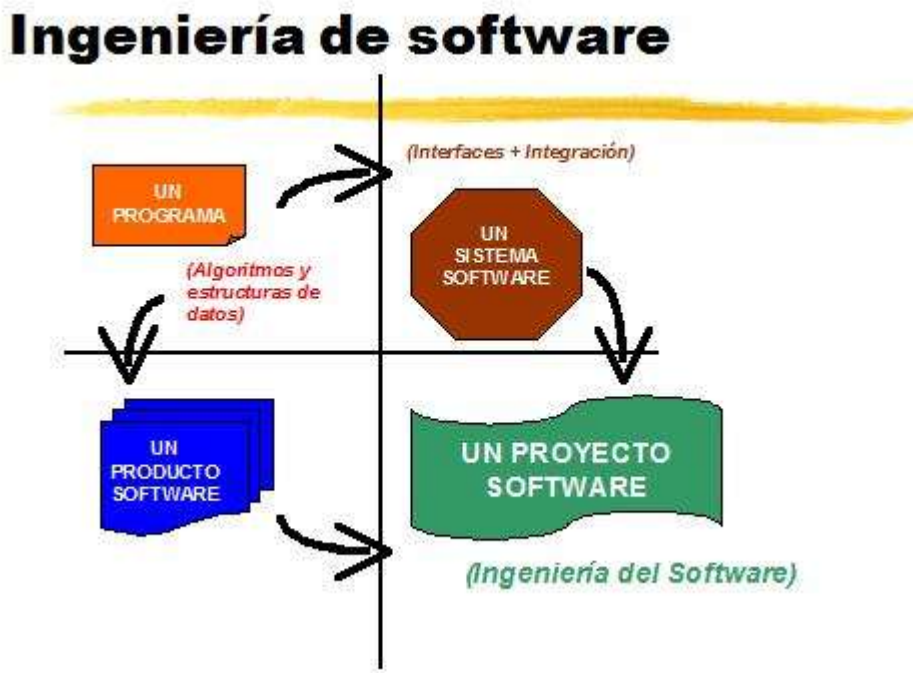
3 **Bauer. Software Engineering.**
Ingeniería del software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable que sea fiable y trabaje en máquinas reales.

4 **Pressman. Ingeniería del Software**
La Ingeniería de/l software es una disciplina o área de la informática o Ciencias de la Computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo.

5 **Braude. Ingeniería de Software**
La ingeniería de software es el proceso de construir aplicaciones de tamaño o alcance prácticos, en las que predomina el esfuerzo del software y que satisfacen los requerimientos de funcionalidad y desempeño.

6 **IEEE**
La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software.

2.2 Esquema de la Ingeniería del Software

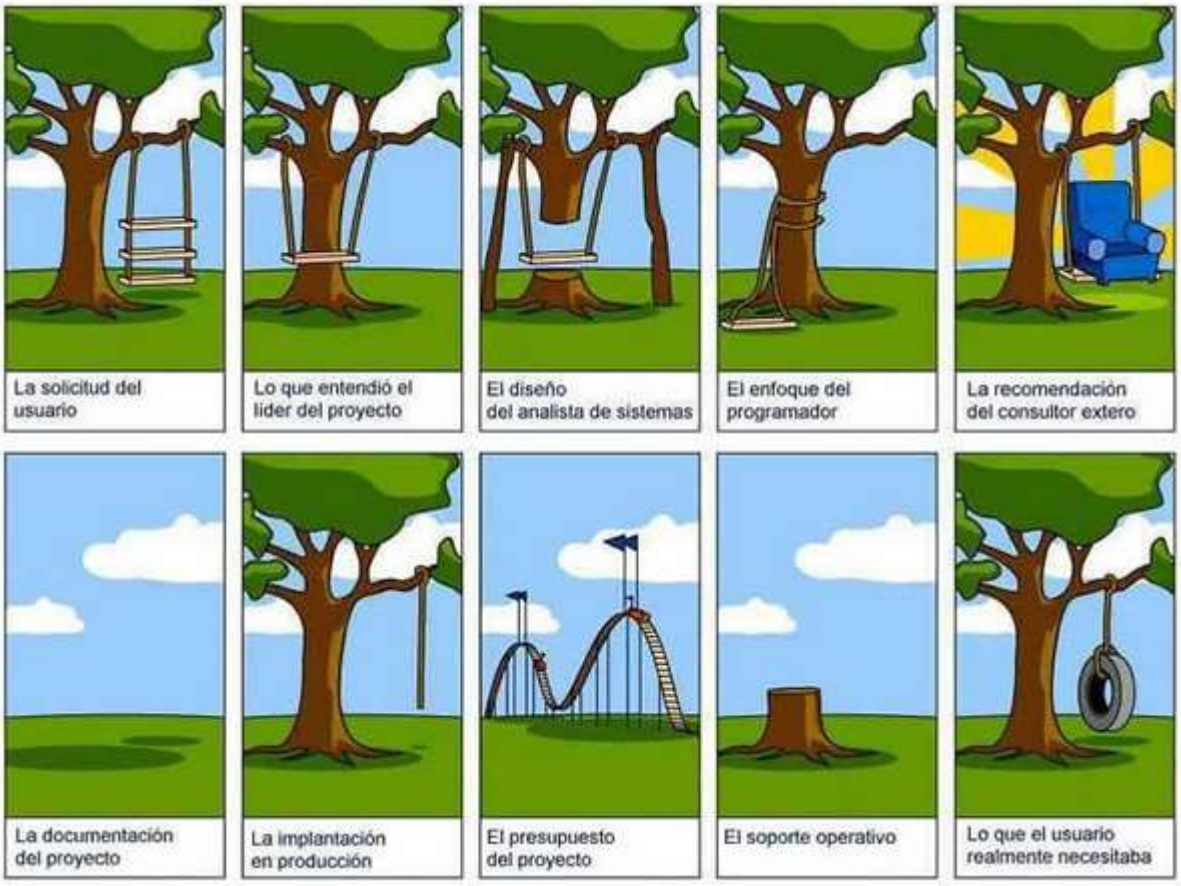


Es muy simple el esquema que consiste en desarrollar un programa sencillo que resuelve una tarea bien determinada. Lo normal es que se evolucione al desarrollo de un

- Sistema software: integra varios programas, o
- Producto software: programa usado en diferentes aplicaciones/entornos

Ambos desarrollos "dan lugar a la Ingeniería del Software": Programas integrados que pueden trabajar en varios entornos.

2.3 Esencia de la Ingeniería del Software



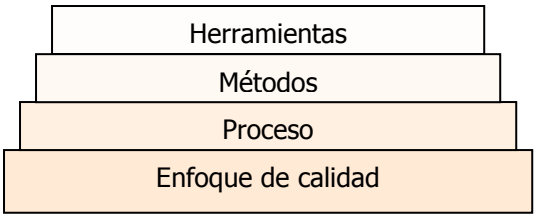
Tomada de http://www.um.es/docencia/barzana/IAGP/IAGP2-Metodologias-de-desarrollo.html

Esta figura podría resumir buena parte de la esencia del curso: en el desarrollo de software (una entidad "compleja") se producen problemas de comunicación a varios niveles: entre usuarios y desarrolladores y entre los componentes mismos del equipo de desarrollo.

Estudiaremos las técnicas, métodos y herramientas de ingeniería que puedan hacer que estos problemas se minimicen, e incluso que desaparezcan.

2.4 Proceso, métodos y herramientas

La ingeniería del software es una tecnología multicapa, y que se apoya sobre un enfoque de calidad.



Enfoque de calidad	Gestión total de calidad. Cultura continua de mejoras de procesos.
Proceso	Define un número de actividades del marco de trabajo aplicables a todos los proyectos del software.
Métodos	Indican "cómo" construir técnicamente el software. Abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento.
Herramientas	Soporte automático o semi-automático para el proceso y los métodos

La ingeniería es el análisis, diseño, construcción, verificación y gestión de entidades técnicas.

El trabajo que se asocia a la ingeniería del software se puede dividir en tres fases, con independencia del área de aplicación, tamaño o complejidad del proyecto.

1

Fase de definición

Se centra sobre el **qué.** Identificar qué información ha de ser procesada, que función y rendimiento se desea, qué comportamiento del sistema, qué interfaces van a ser establecidas, qué restricciones de diseño existen, y qué criterios de validación se necesitan para definir un sistema correcto.
Identificar los requisitos del sistema y del software.
Las tareas específicas de esta fase son:

- Ingeniería de Sistemas o de información
- Planificación del proyecto software
- Análisis de requerimientos

2

Fase de desarrollo

Se centra en el **cómo**. Definir cómo han de diseñarse las estructuras de datos, cómo ha de implementarse la función dentro de una arquitectura de software, cómo ha de implementarse los detalles procedimentales, cómo han de caracterizarse interfaces, cómo ha de traducirse el diseño en un lenguaje de programación y cómo ha de realizarse la prueba.
Las tareas específicas de esta fase son:

- Diseño del software
- Generación de código
- Prueba del software

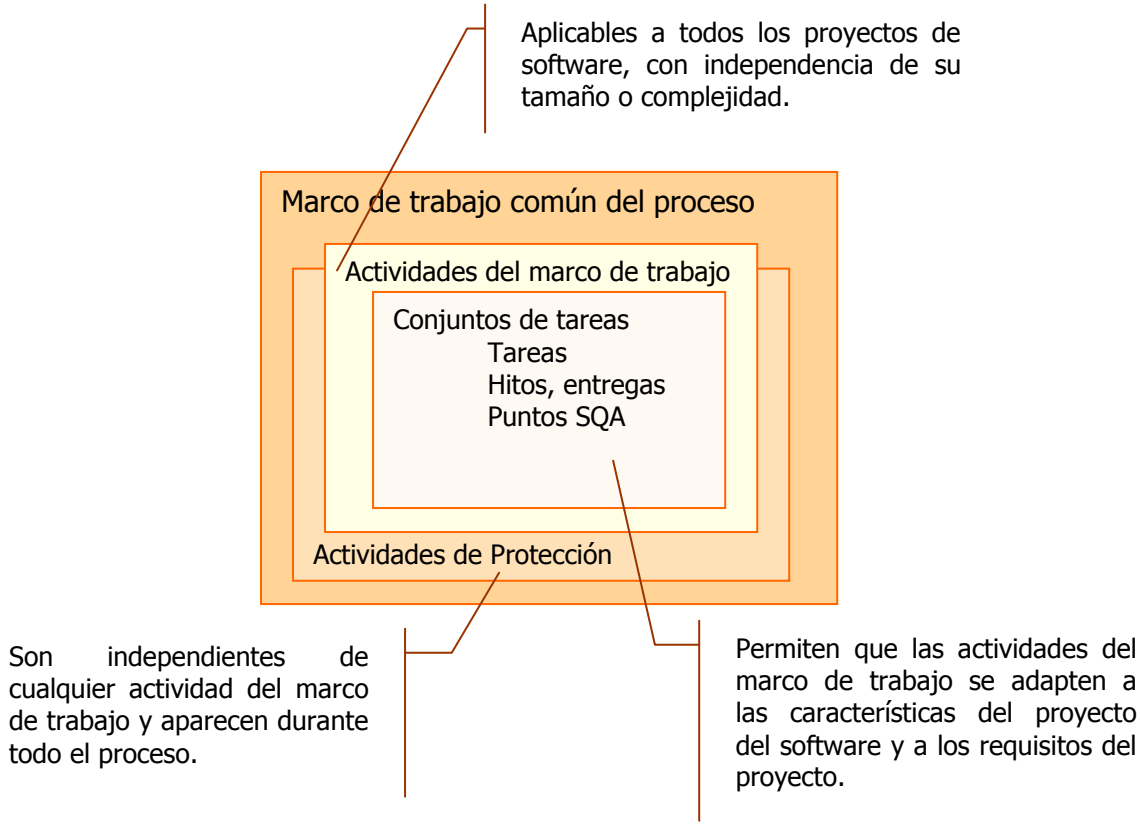
3

Fase de mantenimiento

Se centra en el **cambio**.

- Corrección de errores
- Adaptaciones requeridas a medida que evoluciona el entorno del software
- Cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente
- Se encuentran cuatro tipos de cambio:
  - Corrección
  - Adaptación
  - Mejora
  - Prevención

Un proceso de software se puede caracterizar así:



3. MODELOS DE PROCESO DEL SOFTWARE

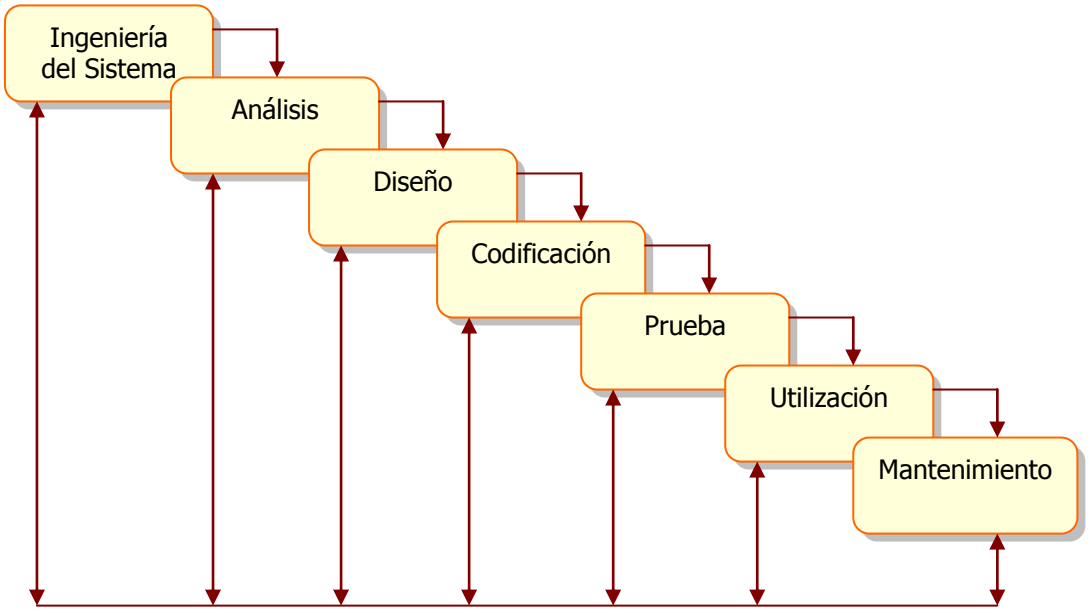
Es importante incorporar estrategias de desarrollo que acompañe al proceso, métodos y a las herramientas.

Una estrategia a menudo de llama *modelo de proceso o paradigma de ingeniería del software*. Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizarse, y los controles y entregas que se requieren.

3.1 El modelo lineal secuencial

Llamado algunas veces “ciclo de vida básico” o “modelo en cascada”, el modelo lineal secuencial sugiere un enfoque sistemático, secuencial, para el desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento.

Es un ciclo de vida en sentido amplio, que incluye no sólo las etapas de ingeniería sino toda la vida del producto: las pruebas, el uso (la vida útil del software) y el mantenimiento.



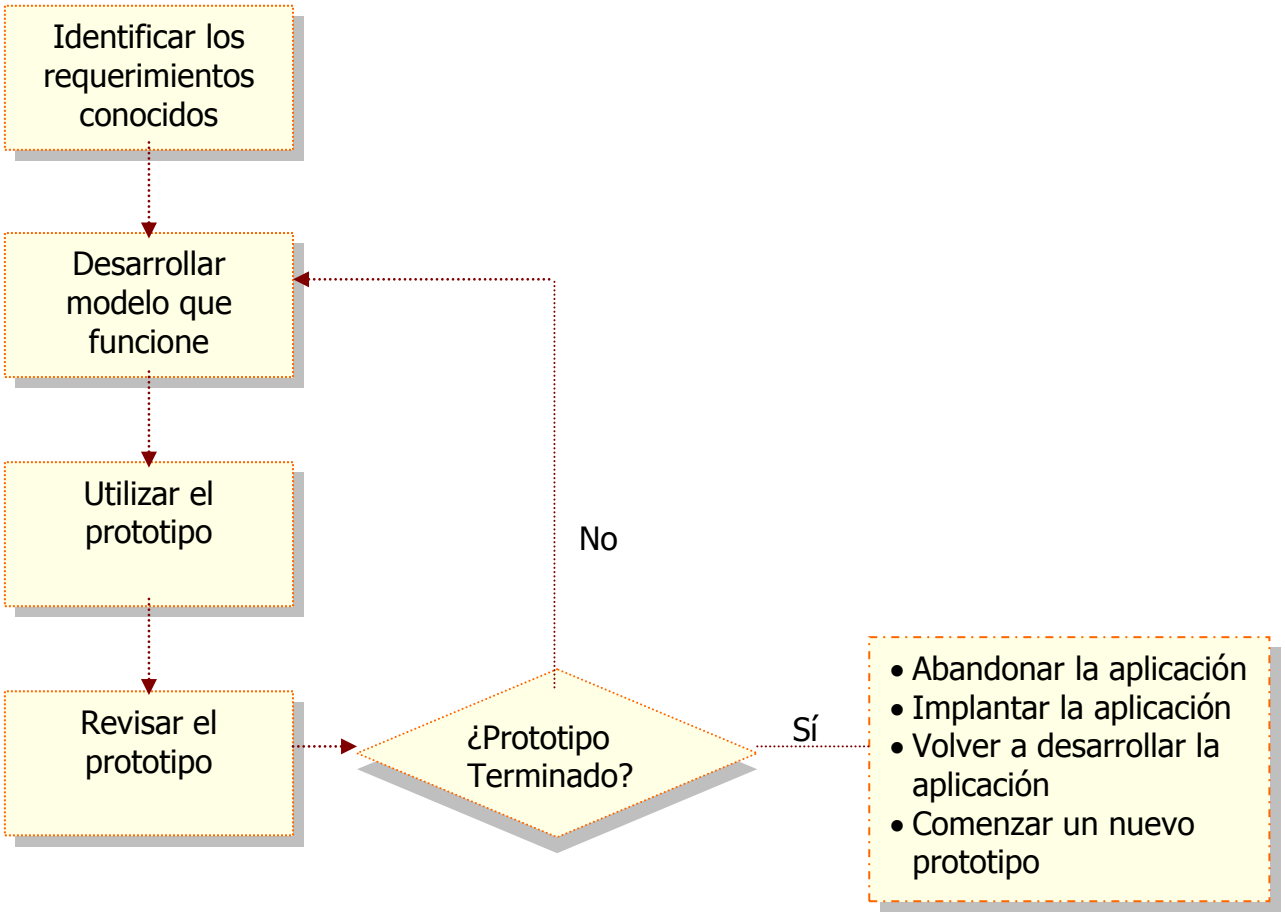
Ingeniería del Sistema	Análisis de las características y el comportamiento del sistema del cual el software va a formar parte.

	Para un sistema nuevo: Se debe analizar cuáles son los requisitos funciones del sistema, y luego asignar un subconjunto de estos requisitos y funciones al software. Para un sistema ya existente: se debe analizar el funcionamiento de la organización y sus operaciones y se asigna al software aquellas funciones que se van a automatizar. Está formado por diagramas y por descripciones en lenguaje natural.

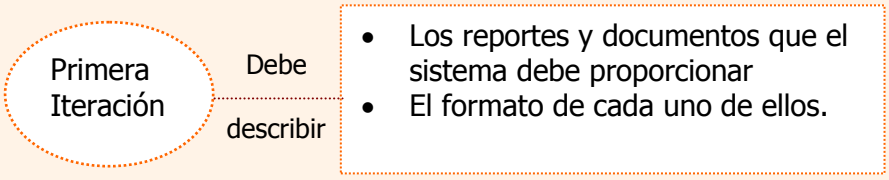
Análisis	Se debe comprender cuáles son los datos que se van a manejar, cuál va a ser la función que tiene que cumplir el software, cuáles son las interfaces requeridas y cuál es el rendimiento y otros requisitos no funcionales que se esperan lograr. Los requisitos, tanto del sistema como del software deben documentarse y revisarse con el cliente. Como resultado del la fase de análisis, se obtiene la especificación de requisitos del software. También está formado por diagramas y descripciones en lenguaje natural.
Diseño	El diseño se aplica a cuatro características distintas del software: la estructura de los datos, la arquitectura de las aplicaciones, la estructura interna de los programas y las interfaces. El diseño es el proceso que traduce los requisitos en una representación del software de forma que pueda conocerse la arquitectura, funcionalidad e incluso la calidad del mismo antes de comenzar la codificación. En el diseño, los requisitos del software se traducen a una serie de diagramas que representan la estructura del sistema software, de sus datos, de sus programas y de sus interfaces.
Codificación	Consiste en la traducción del diseño a un formato que sea comprensible para la máquina. Si el diseño es lo suficientemente detallado, la codificación es relativamente sencilla, y puede hacerse de forma automática, usando generadores de código. Se traducen los diagramas de diseño a un lenguaje fuente, que luego se traduce - se compila - para obtener un programa ejecutable.
Prueba	El objetivo es comprobar que no se hayan producido errores en alguna de las fases anteriores, especialmente en la codificación. Se deben probar todas las sentencias, y todos los módulos que forman parte del sistema.
Utilización	El software se entrega al cliente y comienza la vida útil del mismo.
Mantenimiento	El software sufrirá cambios a lo largo de su vida útil. Estos cambios pueden ser debidos a tres causas:

	Que, durante la utilización, el cliente detecte errores en el software: los errores latentes. Que se produzcan cambios en alguno de los componentes del sistema. Que el cliente requiera modificaciones funcionales no contempladas en el proyecto.

3.2 El modelo de construcción de prototipos



Paso	Descripción
Identificar los requerimientos conocidos	Los analistas y los usuarios trabajan juntos para identificar los requerimientos conocidos que tienen que satisfacerse. Se debe: determinar los fines del sistema y el alcance de su capacidad.

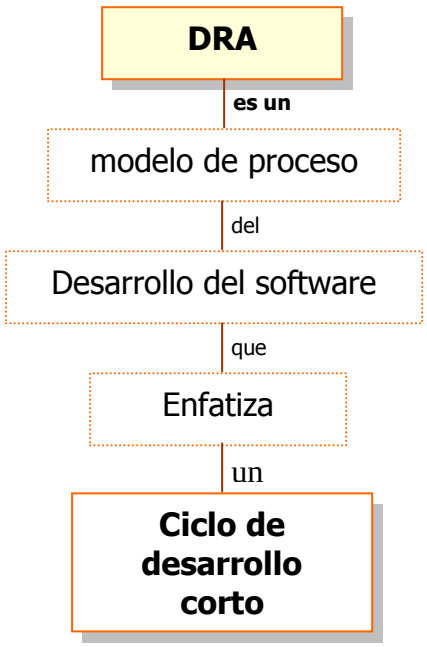
Paso	Descripción
Desarrollar modelo que funcione	Los desarrolladores explican a los usuarios: • El método • Las actividades a realizar • La secuencia en que se llevará a cabo • La responsabilidad de cada participante El proceso de construcción del prototipo se debe iniciar con el desarrollo de un plan general que permita conocer el proceso de desarrollo. Es importante definir un cronograma para el inicio y fin de la primera iteración.  El desarrollador estima los costos asociados con el desarrollo del prototipo. En el desarrollo del prototipo se preparan los siguientes componentes: • El lenguaje de diálogo o conversación entre el usuario y el sistema • Pantallas y formatos para la entrada de datos • Módulos esenciales de procesamiento • Salida del sistema En esta fase no se prepara la documentación ni las especificaciones de salida o de diseño del software.
Utilizar el prototipo	La responsabilidad de trabajar con el prototipo y evaluar sus características y operación es del usuario. La experiencia con el sistema bajo condiciones reales permite determinar los cambios o mejoras o eliminar características innecesarias.

Paso	Descripción
Revisar el prototipo	Se realiza la evaluación y con la información obtenida se levantan las características que debe llevar la siguiente versión del prototipo. La evaluación permite profundizar los rasgos de los usuarios y los de la organización que tienen influencia sobre la aplicación y en su implementación. Los cambios en el prototipo son planificados con los usuarios antes de llevarlos a cabo por el analista.
¿Prototipo terminado?	Los pasos anteriores se repiten varias veces (4 o 6 iteraciones) cuando los usuarios y desarrolladores están de acuerdo en que el sistema ha evolucionado lo suficiente e incluye todas las características necesarias. Cuando el prototipo está terminado, el paso que sigue a continuación es tomar la decisión sobre cómo proceder, para lo cual existen cuatro opciones: **Abandonar la aplicación** Se descartan el prototipo y la aplicación. El desarrollo del prototipo proporcionó información a partir de la cual se determinó que la aplicación o el enfoque seleccionado son inapropiados para justificar un desarrollo adicional. **Implantar el prototipo** Las características y funcionamiento del prototipo satisfacen las necesidades de los usuarios ya sea en forma permanente o para un futuro. **Volver a desarrollar la aplicación** El desarrollo del prototipo proporcionó suficiente información para determinar las características necesarias de toda la aplicación. La información se utiliza como punto de partida para el desarrollo de la aplicación en forma tal haga el mejor uso posible de los recursos. **Comenzar un nuevo prototipo** La información ganada con el desarrollo del prototipo inicial sugiere otras opciones o circunstancia. Se construye un prototipo diferente para añadir información relacionada con los requerimientos de aplicación.

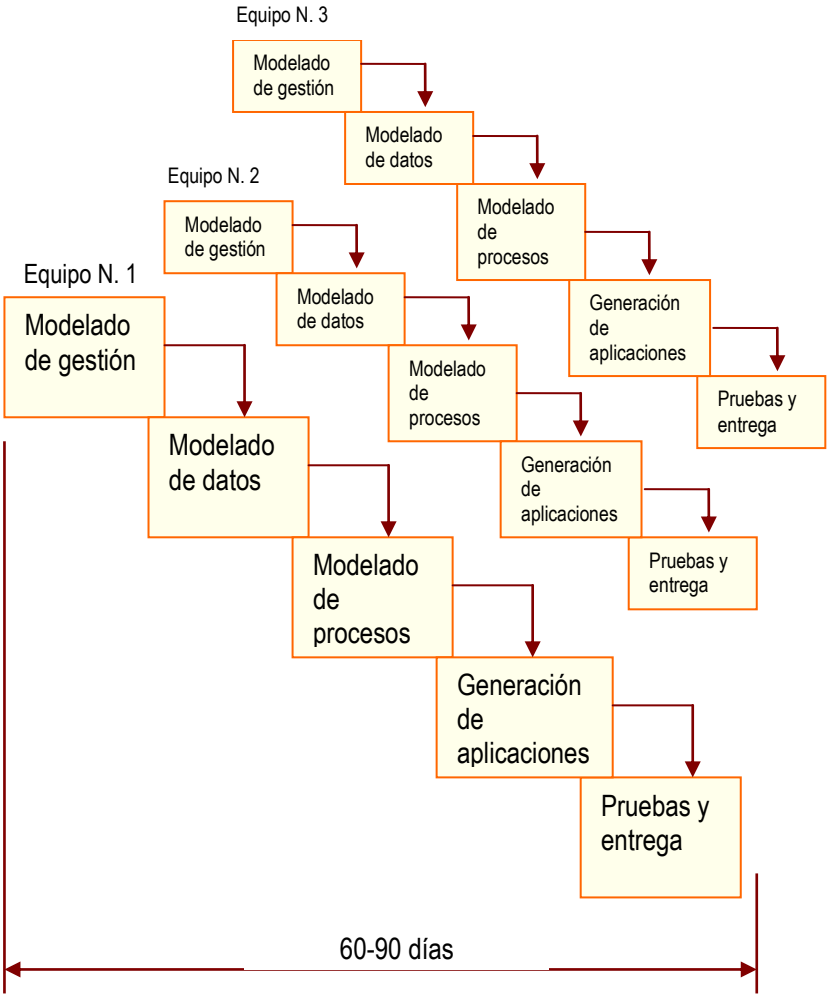
Un prototipo puede tener alguna de las tres formas siguientes:

1. Un prototipo, en papel o ejecutable en computador, que describa la interacción hombre-máquina y los listados del sistema.
2. Un prototipo que implemente algún(os) subconjunto(s) de la función requerida, y que sirva para evaluar el rendimiento de un algoritmo o las necesidades de capacidad de almacenamiento y velocidad de cálculo del sistema final.
3. Un programa que realice en todo o en parte la función deseada pero que tenga características que deban ser mejoradas durante el desarrollo del proyecto.

3.3 El modelo DRA (Desarrollo Rápido de Aplicaciones)



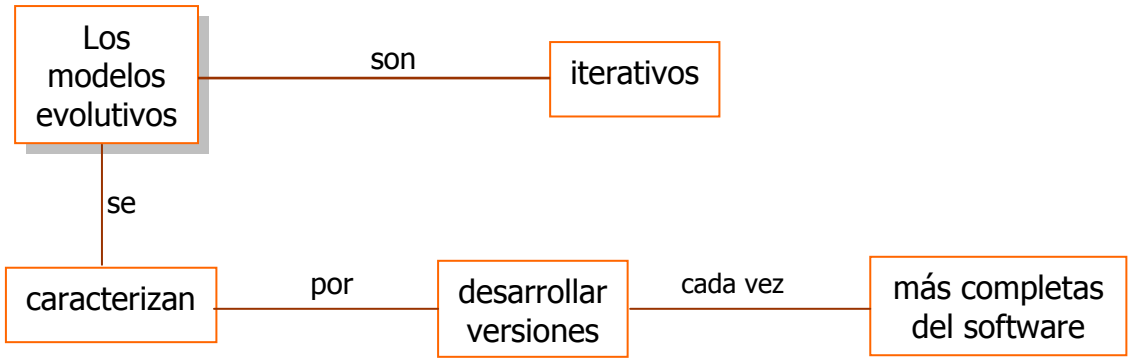
El proceso DRA permite al equipo de desarrollo crear un "sistema completamente funcional" dentro de periodos cortos de tiempo (de 60 a 90 días). El enfoque DRA comprende las siguientes fases:



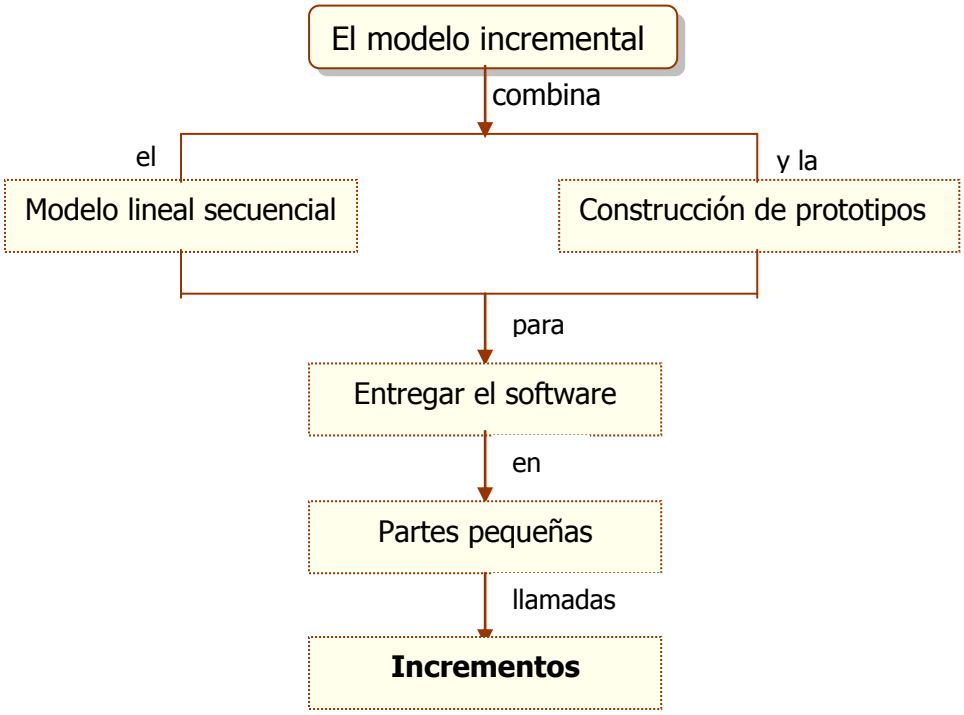
Modelado de gestión	El flujo de información entre las funciones de gestión se modela de forma que responda a las siguientes preguntas: ¿Qué información conduce al proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la procesa?

Modelado de datos	Conjunto de objetos de datos necesarios para apoyar la empresa. Se definen las características (atributos) de cada uno de los objetos y las relaciones entre estos objetos.
Modelado del proceso	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.
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 o crear componentes reutilizables.
Pruebas y 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.

3.4 Modelos de procesos evolutivos de software

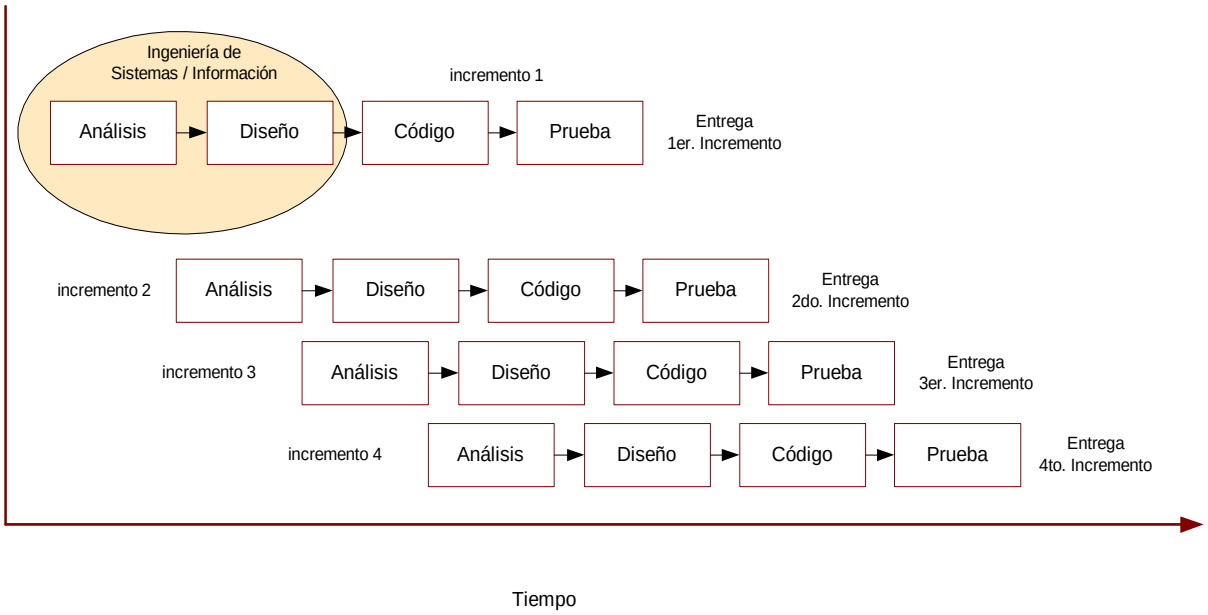


3.4.1 El modelo incremental

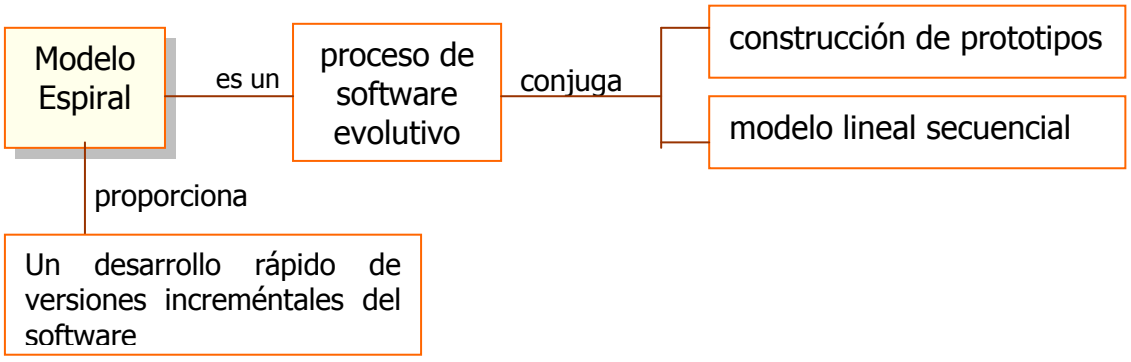


El modelo incremental se centra en la entrega de un producto operacional con cada incremento.

Los primeros incrementos son versiones "incompletas" del producto final, pero proporcionan al usuario la funcionalidad necesaria para su evaluación.



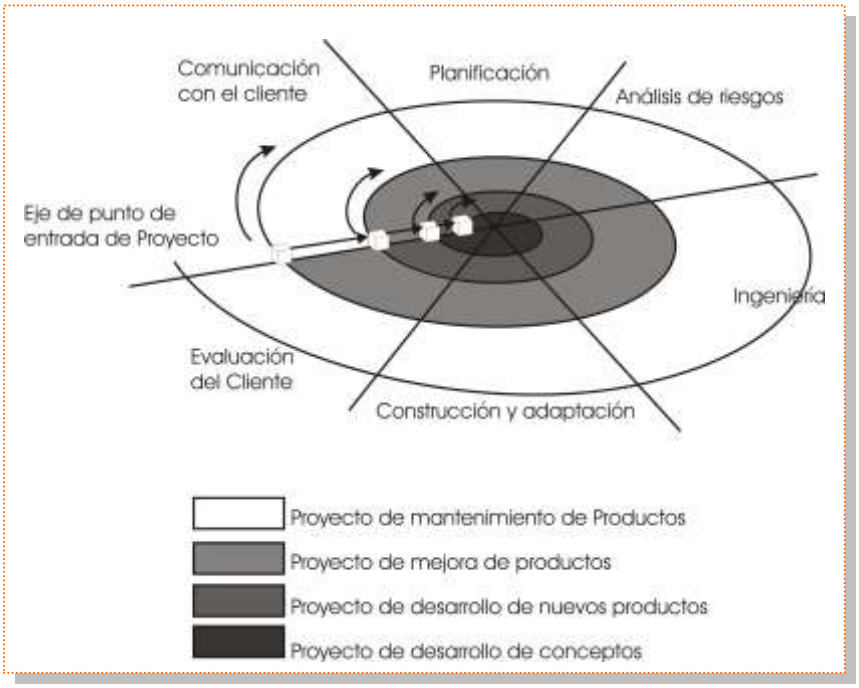
3.4.2 El modelo espiral



En las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo.
Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.

Actividades del modelo en espiral

- **Comunicación con el cliente:** Se establece comunicación entre el desarrollador y el cliente.
- **Planificación:** Se definen los recursos, el tiempo y otra información relacionados con el proyecto.
- **Análisis de riesgos:** Se evalúan riesgos técnicos y de gestión
- **Ingeniería:** Se construyen una o más representaciones de la aplicación.
- **Construcción y acción:** Construir, probar, instalar y proporcionar soporte al usuario.
- **Evaluación del cliente:** Se obtiene la reacción del cliente. Se realiza la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.



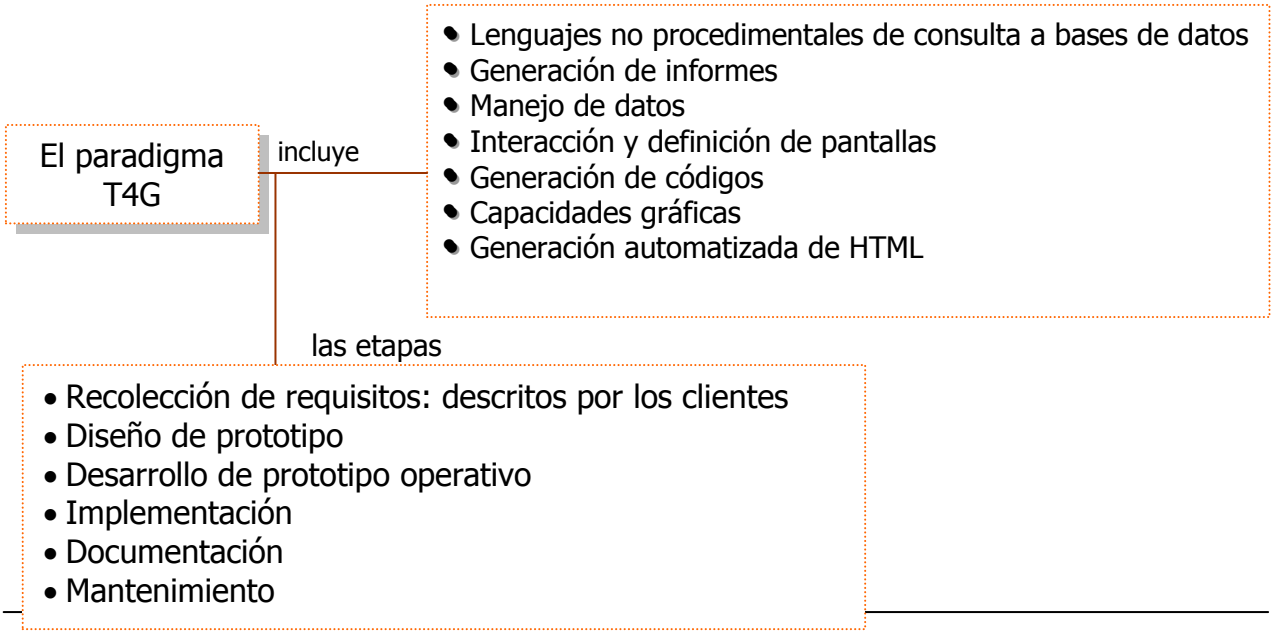
El equipo de ingeniería del software gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro.

El primer circuito de la espiral puede producir el desarrollo de una especificación de productos; los pasos siguientes en la espiral se podrían utilizar para desarrollar un prototipo y progresivamente versiones mas sofisticadas del software. Cada paso por la región de planificación produce ajustes en el plan del proyecto. El costo y la planificación se ajustan con la realimentación ante la evaluación del cliente. Además, el gestor del proyecto ajusta el número planificado de iteraciones requeridas para completar el software.

3.5 Modelo de métodos formales y Técnicas de cuarta generación

El modelo de métodos formales comprende un conjunto de actividades que conducen a la especificación matemática del software de computadora. Los métodos formales permiten que un ingeniero de software especifique, desarrolle y verifique un sistema basado en computadora aplicando una notación rigurosa y matemática.
Este enfoque es llamado *ingeniería del software de sala limpia*. Cuando se utilizan métodos formales se descubren y corrigen ambigüedades, inconsistencias y errores.

Las técnicas de cuarta generación, abarcan un conjunto de herramientas que facilitan al ingeniero del software la especificación de las características del software a alto nivel.
La herramienta genera automáticamente el código fuente basándose en la especificación del técnico. Cuanto mayor sea el nivel en el que se especifique el software, más rápido se puede construir el programa.
El paradigma T4G para la ingeniería del software se orienta hacia la posibilidad de especificar el software usando formas de lenguaje especializado o notaciones gráficas que describa el problema que hay que resolver en términos que los entienda el cliente.



ACTIVIDADES COMPLEMENTARIAS

1. Las capas de la Ingeniería del Software sitúa las tres capas encima de la capa titulada “enfoque de calidad”. Esto implica un programa de calidad tal como Gestión de Calidad Total. Investigue y desarrolle un esquema de los principios clave de un programa de Gestión de Calidad Total.

2. Elabore y proporcione una tabla donde se especifiquen las ventajas y desventajas de los diferentes paradigmas de ingeniería de software.

3. ¿Qué paradigmas de ingeniería del software de los presentados piensa que sería el más eficaz? ¿Por que?

4. ¿Qué es más importante, el producto o el proceso?

BIBLIOGRAFIA

IMPRESA

BRAUDE. Ingeniería de software, una perspectiva orientada a objetos. México. 2003. Alfaomega grupo editor. S.A.

GRUEGGE, BERND y DUTOIT, Allen H. Ingeniería de software orientado a objetos. México. 2002. Pearson Educación.

HUMPHREY, Watts S. Introducción al proceso de software personal. Pearson Addison wesley. 2001.

MEYER, Bertrand. Construcción de software orientado a objetos. Segunda edición. Madrid. 1999. Prentice Hall.

NORRIS. Ingeniería de software explicada. Grupo Noriega editores de Colombia.

PIATTINI, Mario. VILLALBA, José y otros. Mantenimiento del software: modelos, técnicas y métodos para la gestión del cambio. Editorial Alfaomega-Rama.

PRESSMAN, Roger S. Ingeniería del Software. Un enfoque práctico. Quinta edición. España. 2002. Editorial McGraw Hill.

PFLEEGER, Shari Lawrence. Ingeniería de software, teoría y práctica. 1ª. Edición. Buenos Aires. Pearson educación. 2002

SOMMERVILLE, Ian. Ingeniería de software. 6ª. Edición. Pearson Addison Wesley. 2001

UNIDAD 2.
GESTIÓN Y PLANIFICACIÓN DE PROYECTOS SOFTWARE

INTRODUCCIÓN

La gestión y planificación de proyectos es una actividad que empieza antes de iniciar cualquier actividad técnica y continúa a lo largo de la definición, del desarrollo y del mantenimiento del software.

La actividad de gestión del proyecto comprende medición y métricas, estimación, análisis de riesgos, planificación, seguimiento y control.

OBJETIVOS

GENERALES

- Estudiar las técnicas de gestión necesarias para planificar, organizar, supervisar y controlar proyectos de software.
- Estudiar las técnicas de análisis y gestión del riesgo.
- Estudiar las técnicas de gestión necesarias para planificar proyectos de software.

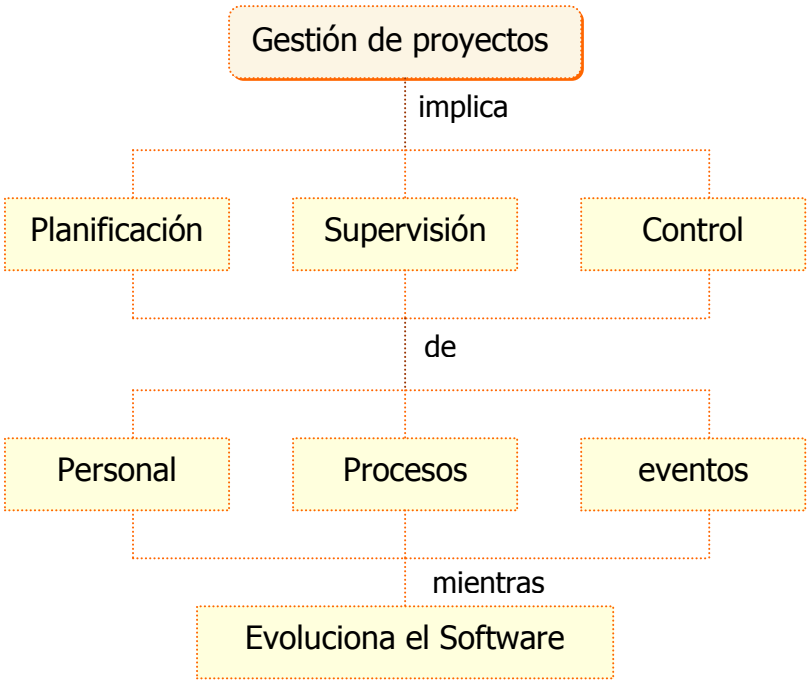
ESPECIFICOS

- Determinar como se debe gestionar el personal, el proceso y el problema durante un proyecto de software.
- Identificar las métricas de software y cómo pueden emplearse para gestionar el proceso de software y el proyecto llevado a cabo como parte del proceso.
- Determinar como se crea la planificación temporal de un proyecto.
- Identificar la garantía de calidad del software.
- Determinar los riesgos del software.
- Identificar los riesgos del software.
- Determinar la proyección y evaluación del riesgo.

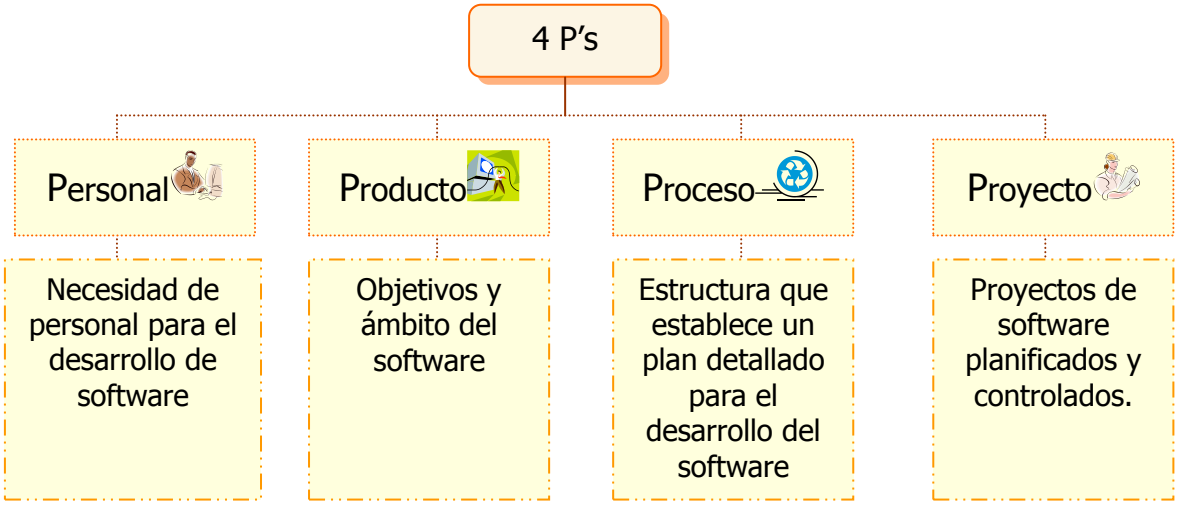
ESTRUCTURA TEMÁTICA

1. CONCEPTOS SOBRE GESTION DE PROYECTOS

1.1 Gestión de proyectos



La gestión de un proyecto de software se centra en:



1.2 Personal
Recurso humano que participa y colabora en el proceso del software y su organización para el desarrollo de los proyectos software de manera eficaz.

Participantes	Se clasifican en: 1. **Gestores Superiores**: se encargan de definir los aspectos del negocio. 2. **Gestores técnicos del proyecto**: se encargan de planificar, motivar, organizar y controlar a los profesionales que realizan el trabajo de desarrollo del software. 3. **Profesionales**: se encargan de proporcionan las capacidades técnicas necesarias para la ingeniería de un producto o aplicación. 4. **Clientes**: especifican los requisitos para la ingeniería del software. 5. **Usuarios finales**: Se encargan de interactuar con el software.
Jefes de equipo	Es el gestor de proyectos de software, el cual: • Diagnostica los aspectos técnicos y de organización más relevantes. • Tiene confianza para asumir el control del proyecto y permite que los buenos técnicos aporten sus ideas. • Promueve e incentiva las iniciativas y logros del equipo del proyecto. • Hace saber a todos los miembros del equipo que la calidad es importante.
Equipo de software	Mantei, propone 3 niveles de organización de equipos. **Descentralizado democrático** Este equipo no tiene un jefe permanente y se nombran coordinadores a corto plazo. Las decisiones se hacen por consenso del grupo. La comunicación entre los miembros del equipo es horizontal.

	Descentralizado controlado Este equipo tiene un jefe definido que coordina tareas específicas y jefes secundarios que tienen responsabilidades sobre subtareas. La resolución de problemas sigue siendo una actividad del grupo, pero la implementación de soluciones se reparte entre subgrupos por el jefe de equipo. La comunicación entre subgrupos e individuos es horizontal. También hay comunicación vertical a lo largo de la jerarquía de control. **Centralizado controlado** El jefe del equipo se encarga de la resolución de problemas a alto nivel y la coordinación interna del equipo. La comunicación entre jefe y los miembros del equipo es vertical.
Coordinación y Comunicación	Se establecen mecanismos de comunicación para coordinar al equipo de trabajo. Se deben tener: **Comunicación formal:** se lleva a cabo por escrito, con reuniones organizadas y otros canales de comunicación. Incluye documentos de ingeniería de software, memorandos técnicos, documentación, informes de seguimiento. **Comunicación informal:** es más personal. Incluye reuniones de grupo para la divulgación de información y para la resolución de problemas. **Comunicación electrónica:** se leva a cabo por correos electrónicos, boletines, audioconferencias, videoconferencias.

1.3 Producto

Al inicio de un proyecto, el gestor del proyecto debe examinar el producto y el problema a resolver. Por lo que se debe establecer el ámbito del producto – delimitarlo.

Ámbito	Se define: • **Contexto**: ¿Cómo encaja el software a construir en un sistema, producto o contexto de negocios mayor y qué limitaciones se imponen como resultado del contexto? • **Objetivos de información**: ¿Qué objetos de datos visibles al cliente se obtienen del software? ¿Qué objetos de datos son requeridos de entrada? • **Función y rendimiento**: ¿Qué función realiza el software para transformar la información de entrada en una salida? ¿Hay características de rendimiento especiales que abordar?
Descomposición del problema	Comprende el análisis de requisitos del software. La descomposición se aplica en dos áreas principales: (1) la funcionalidad que debe entregarse y (2) el proceso que se empleará para entregarlo. Un problema complejo se parte en problemas más pequeños que resultan más manejables.

1.4 Proceso

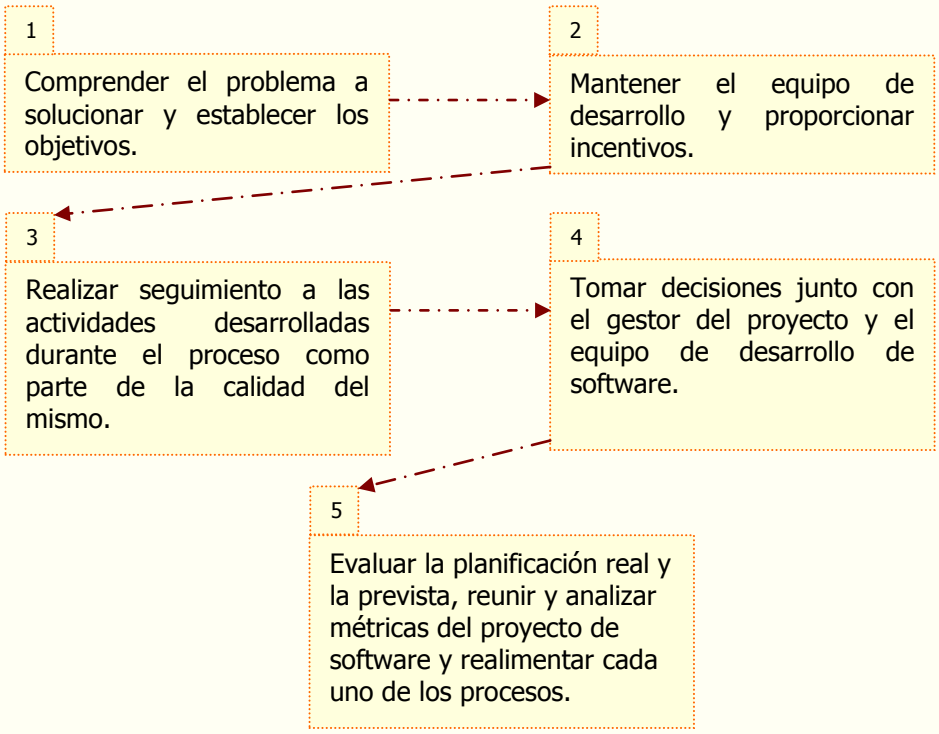
El gestor del proyecto decide qué modelo de proceso es el más adecuado para:
1. Los clientes que han solicitado el producto y la gente que realizará el trabajo.
2. Las características del producto.
3. El entorno del proyecto.

Maduración del problema y el proceso	Los miembros del equipo de software deben estructurar un conjunto de actividades que le permitan trabajar en cada función del problema. Se pueden considerar las siguientes actividades: **Comunicación** Se establece comunicación entre el desarrollador y el cliente, con el propósito de obtener los requisitos del sistema. **Planificación** Conjunto de tareas con el propósito de definir los recursos y la planificación temporal del proyecto. **Análisis del riesgo** Tareas requeridas para valorar los riesgos técnicos y de gestión. **Ingeniería** Tareas requeridas para construir una o más representaciones de la aplicación. **Construcción y entrega** Tareas requeridas para construir, probar, instalar y proporcionar asistencia al usuario. **Evaluación del cliente** Tareas requeridas para que el cliente evalúe las representaciones de software creadas durante la fase de ingeniería. recursos, poner fechas de inicio y finalización de las tareas y los productos a fabricar.
Descomposición del proceso	Las actividades de: comunicación, planificación, análisis de riesgo, ingeniería, construcción, entrega y evaluación se adaptan al modelo o paradigma de desarrollo de software seleccionado.

1.5 Proyecto

Se deben gestionar proyectos software de calidad para que tengan éxito.

Se debe:

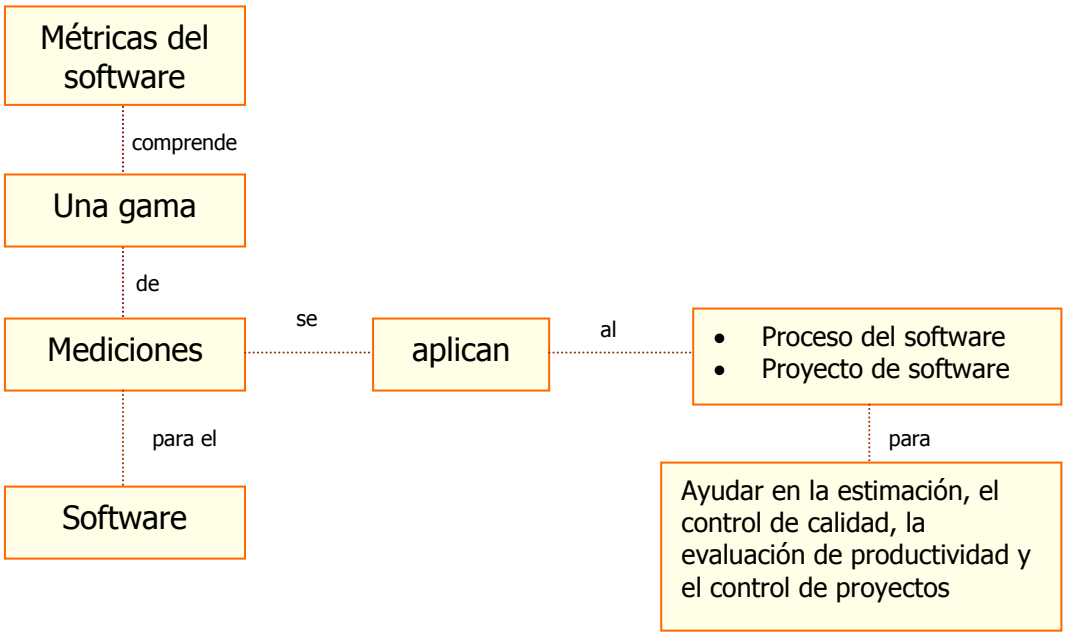


ACTIVIDADES COMPLEMENTARIAS

1. Se le ha nombrado gestor de proyecto dentro de una organización de sistemas de información. Su trabajo es construir una aplicación que es bastante similar a otras que ha construido su equipo, aunque ésta es mayor y más compleja. Los requisitos han sido detalladamente documentados por el cliente. ¿Qué estructura de equipo elegiría y porqué? ¿Qué modelo(s) de proceso de software elegiría y por qué?

2. Se le ha nombrado gestor de proyecto de una pequeña compañía de productos software. Su trabajo consiste en construir un producto innovador que combine hardware de realidad virtual con software innovador. Puesto que la competencia por el mercado de entretenimiento casero es intensa, hay cierta presión para terminar el trabajo rápidamente. ¿Qué estructura de equipo elegiría y porqué? ¿Qué modelo(s) de proceso de software elegiría y por qué?

2. EL PROCESO DE SOFTWARE Y MÉTRICAS DEL PROYECTO



Las razones para medir los procesos del software, los productos y los recursos: son:

- **Caracterizar**: para comprender mejor los procesos, los productos, los recursos y los entornos
- **Evaluar**: para determinar el estado con respecto al diseño
- **Predecir**: para poder planificar
- **Mejorar**: la calidad del producto y el rendimiento del proceso.

2.1 Métricas en el proceso y dominios del proyecto

Dentro de la Ingeniería del software se manejan los siguientes conceptos:

Medida: Proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto.

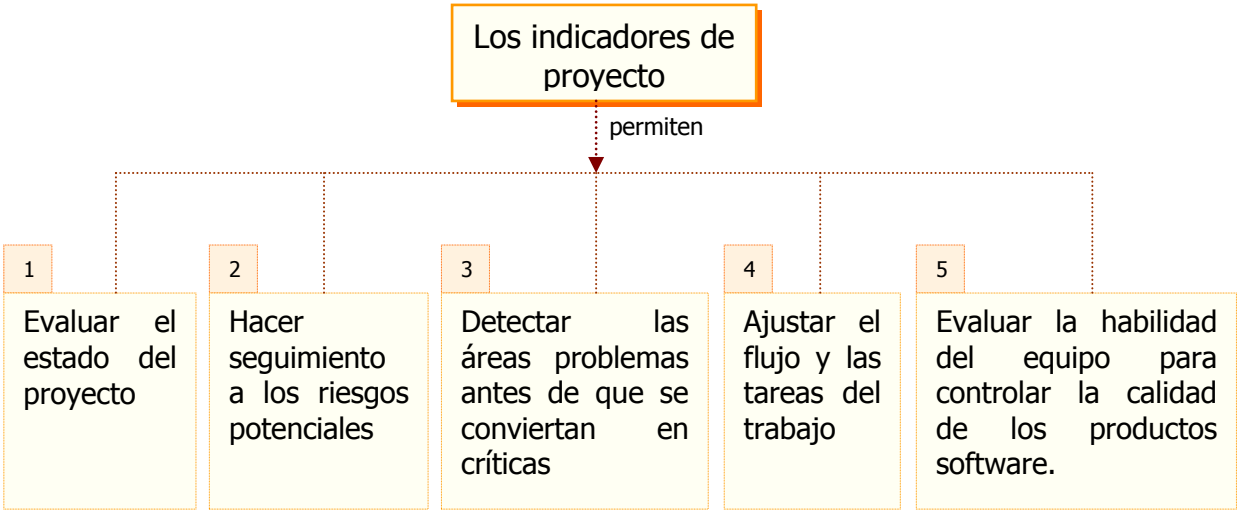
Medición: es el acto de determinar una medida.

Métrica: Una medida cuantitativa del grado en que el sistema, componente o proceso posee un atributo dado.

Indicador: es una métrica o una combinación de métricas que proporcionan una visión profunda del proceso del software, del proyecto de software o del producto en sí. Un indicador proporciona una visión profunda que permite al gestor de proyectos o a los ingenieros de software ajustar el producto, el proyecto o el proceso.

El objetivo principal de los indicadores de proceso es evaluar las condiciones de funcionamiento de un proceso y poder tener una visión de la eficacia de un proceso existente.

Durante un tiempo considerable se recopilan las métricas de todos los proyectos y se proporcionan los indicadores para obtener mejoras en el software.



Para mejorar cualquier proceso se debe:

- √ Medir atributos del proceso
- √ Definir y desarrollar un juego de métricas para esos atributos
- √ Utilizar las métricas para encontrar indicadores para la estrategia de mejora

De acuerdo a la figura:

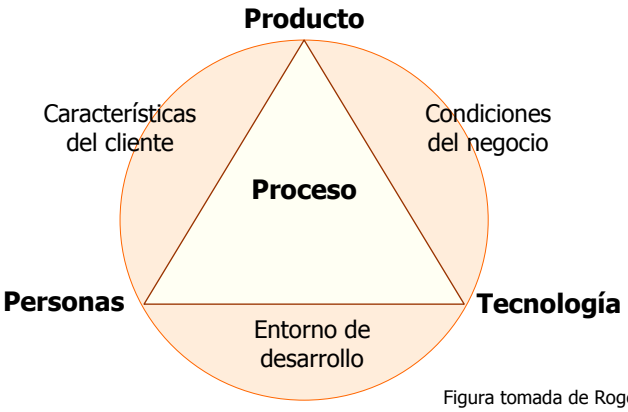


Figura tomada de Roger Presuman. Ingeniería de Software

El producto, la tecnología y las personas tienen una fuerte influencia en el desarrollo y la calidad del software. El proceso se encuentra dentro de unas condiciones de entorno que incluyen: entornos de desarrollo, condiciones del negocio, y características del cliente. Estas condiciones, son de gran importancia puesto que permiten definir las reglas del proceso y poder contribuir a la calidad del software.

La eficacia de un proceso de software se mide a través de un juego de métricas según los resultados que provienen del proceso.

Dentro de éstos resultados se debe incluir:

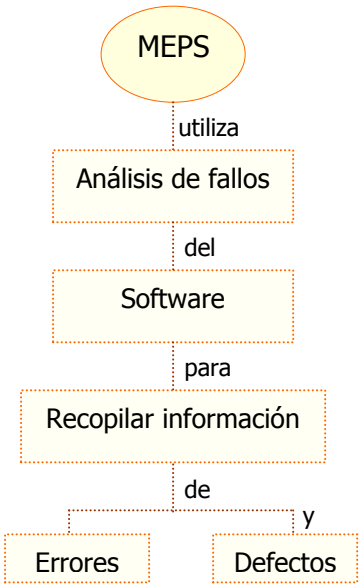
- √ Medida de errores detectados antes de la entrega del software
- √ Defectos detectados
- √ Productos de trabajo entregados
- √ Esfuerzo humano y tiempo consumido
- √ Ajuste con la planificación

También se debe incluir métricas para medir las características de tareas específicas de la ingeniería del software.

- √ Medida del tiempo y del esfuerzo para llevar a cabo actividades de protección
- √ Actividades genéricas de ingeniería del software

2.2 Mejora estadística del proceso del software (MEPS)

Para una organización es importante estar a gusto con la recopilación y la utilización de métricas de proceso, de éstas se deriva la identificación de indicadores llevando a un enfoque más riguroso denominado "Mejora estadística de proceso del software (MEPS)".

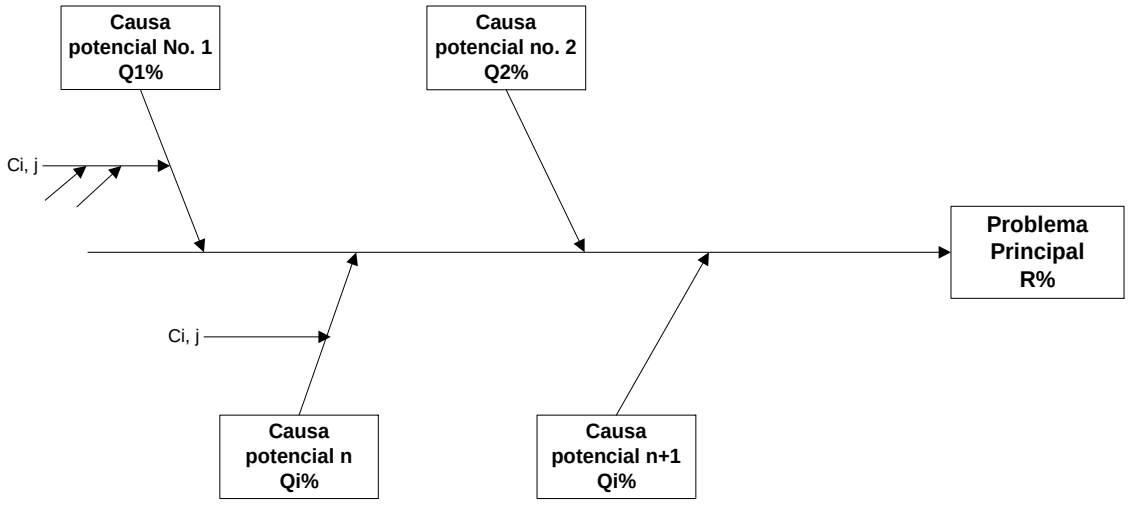


Para realizar un análisis de fallos se deben seguir los siguientes pasos:

1. Categorizar por origen, todos los errores y defectos.
2. Registrar el costo de corregir cada error y el del defecto
3. Contar el número de errores y de defectos de cada categoría y se ordenar por orden descendente
4. Computar el costo global de errores y defectos de cada categoría.
5. Los datos resultantes se analizan para detectar las categorías que producen un

Error	Es alguna fisura descubierta por los ingenieros del software antes de que el software sea entregado al usuario final
Defecto	Es alguna fisura descubierta después de la entrega del software al usuario final

Para determinar las principales causas que pueden ocasionar defectos en el software y con base en ello extraer los indicadores que permitan a una organización de software modificar su proceso para reducir la frecuencia de errores y defectos se utiliza el diagrama de espina.



Ci, j : Causa asociada a cada subproblema

Qi %: Porcentaje de relevancia del subproblema

R% : Porcentaje de relevancia del problema principal

En un diagrama de espina:

- La línea central, representa el factor de calidad o el problema en consideración.
- Las líneas diagonales conectadas a la línea central indican una causa potencial del problema de calidad.

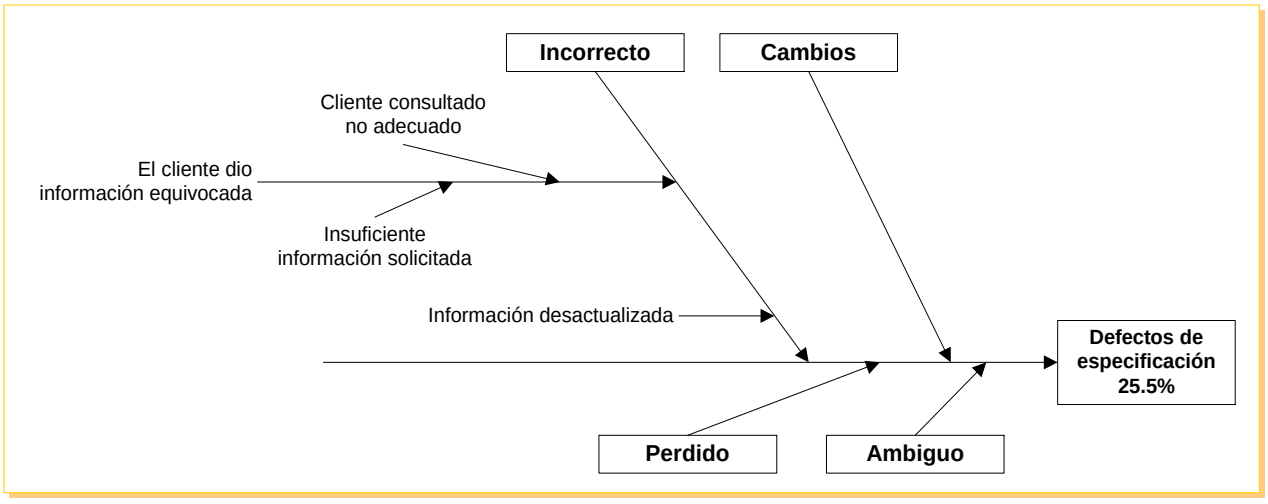
Esta misma notación se aplica para cada una de las líneas diagonales conectadas a la línea central.

Por ejemplo:

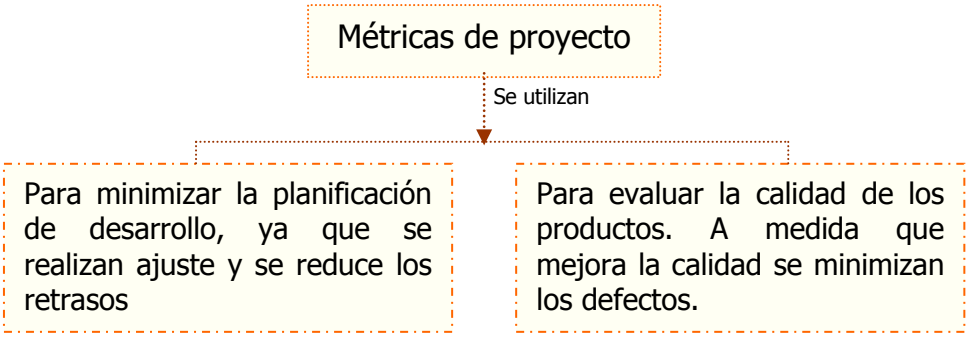
Se han encontrado y determinado las siguientes causas y su origen en un proyecto de software:

Origen de errores / defectos	Causa	%
Especificación / requisitos	Lógica	20
	Manejo de datos	10.9
	estándares	6.9
Diseño	Especificaciones	25.5
Código	Interfaz software	6.0
	Interfaz hardware	7.7
	Comprobación de errores	10.9
	Interfaz de usuario	11.7

Si tomamos la causa Especificaciones y utilizamos un diagrama de espina para identificar las causas específicas para este problema, tenemos:



2.3 Métricas del Proyecto



Las métricas del proyecto de software sugieren que los proyectos deben medir:

1. Entradas: la dimensión de los recursos que se requieren para realizar el trabajo
2. Salidas: medidas de las entradas o productos creados durante el proceso de ingeniería del software
3. Resultados: medidas que indican la efectividad de las entregas.

2.4 Mediciones del Software

Las métricas del software se pueden categorizar en:

Medidas directas:
Dentro de estas se pueden incluir:

- El costo y el esfuerzo aplicado
- Líneas de código producidas (LCD)
- Velocidad de ejecución, tamaño de memoria y los defectos informados durante un periodo de tiempo establecido

Medidas Indirectas:
Incluyen:

- La funcionalidad, calidad, complejidad, eficiencia
- fiabilidad, facilidad, facilidad de mantenimiento

El domino de las métricas del software se dividen en métricas de proceso, proyecto y producto.

2.4.1 Métricas orientadas al tamaño

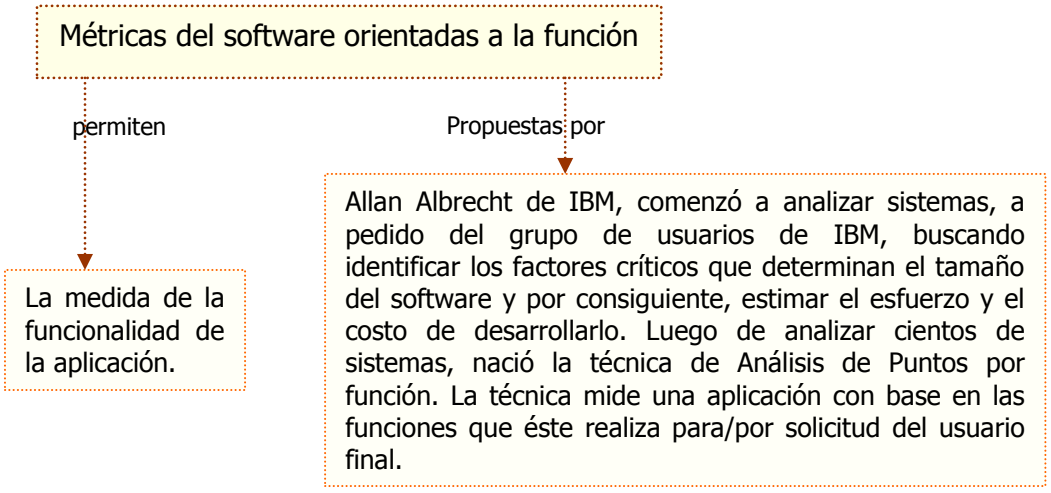
Provienen de la normalización de las medidas de calidad y/o productividad considerando el "tamaño" del software que se haya producido.
Los datos que se deben tener en cuenta, se pueden llevar en la siguiente tabla:

Proyecto	LDC	Esfuerzo	Costo $	Páginas de documentación	Errores	Defectos	Personas
IRIS	18.200	24	200.000	945	134	86	4

Teniendo en cuenta los datos de la tabla, se pueden derivar otras métricas para comparar varios proyectos. Por ejemplo:

- Errores por KLDC (miles de líneas de código)
- Defectos por KLDC
- Páginas de documentación por KLDC
- Errores por persona-mes
- LDC por persona-mes
- Costo ($) por página de documentación

2.4.2 Métricas orientadas a la función



Los puntos de función se obtienen utilizando una función empírica basado en medidas cuantitativas del dominio de información del software y valoraciones subjetivas de la complejidad del software.

Los puntos de función se calculan utilizando la siguiente tabla:

Parámetros de medición	**Cuenta**		**Factor de ponderación**				
			Simple	**Medio**	**Complejo**		
Número de entradas de usuario		X	3	4	6	=	
Número de salidas de usuario		X	4	5	7	=	
Número de peticiones de usuario		X	3	4	6	=	
Número de archivos		X	7	10	15	=	
Número de interfaces externas		X	5	7	10	=	
						Cuenta_total	

Se determinan 5 características del ámbito de la información y los cálculos aparecen en la posición apropiada de la tabla. Los valores del ámbito de información están definidos de la siguiente manera:

1. Número de entradas de usuario: se cuenta cada entrada de usuario que proporcione al software diferentes datos orientados a la aplicación.

2. Número de salidas de usuario: se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto las salidas se refieren a informes, pantallas, mensajes de error.

3. Número de peticiones de usuario: una petición esta definida como una entrada interactiva que resulta de la generación de algún tipo de respuesta en forma de salida interactiva. Se cuenta cada petición por separado.

4. Número de archivos: se cuenta cada archivo maestro lógico.

5. Número de interfaces externas: se cuentan todas las interfaces legibles por la maquina por ejemplo: archivos de datos, en cinta o discos que son utilizados para transmitir información a otro sistema.

Cuando han sido recogidos los datos anteriores, se asocia el valor de complejidad a cada cuenta. Las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada es denominada simple, media o compleja. No obstante la determinación de la complejidad es algo subjetivo.

Para calcular los puntos de función se utiliza la siguiente relación:

$$PF = \textbf{Cuenta\_total} * [0.65 + 0.01 * \sum(f_i)]$$

PF	Punto de función
Cuenta_total	Es la suma de todas las entradas obtenidas
f_i	Donde i=1 hasta 14. Son valores de ajuste de la complejidad basados en las respuestas a las cuestiones señaladas de la siguiente tabla: Evaluar cada factor en escala 0 a 5

0	1	2	3	4	5
No influencia	Incidental	Moderado	Medio	Significativo	Esencial

F_i :	
1	¿Requiere el sistema copias de seguridad y de recuperación fiables?
2	¿Se requiere comunicación de datos?
3	¿Existen funciones de procesamiento distribuido?
4	¿Es crítico el rendimiento?
5	¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado?
6	¿Requiere el sistema entrada de datos interactiva?
7	¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
8	¿Se actualizan los archivos maestros de forma interactiva?
9	¿Son complejas las entradas, las salidas, los

	archivos o las peticiones?
10	¿Es complejo el procesamiento interno?
11	¿Se ha diseñado el código para ser reutilizable?
12	¿Están incluidas en el diseño la conversión y la instalación?
13	¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14	¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Una vez calculado los puntos de función se usan como medida de la productividad, calidad y otros productos del software.

Productividad = PF / persona-mes
Calidad = Errores / PF
Costo = Dólares / PF
Documentación = Paginas Documentadas / PF

2.5 Métricas para la calidad del software

El objetivo de la ingeniería del software es desarrollar y producir software de alta calidad. Para lograr este objetivo, es fundamental aplicar métodos y herramientas efectivos dentro del contexto de un proceso maduro de desarrollo de software.

2.5.1 Medidas de la Calidad

Dentro de las medidas de calidad del software tenemos:

Corrección

Es el grado en que el software cumple su función.
La medida más común es:

Defectos por KDLC (miles de líneas de código)

Facilidad de mantenimiento

Es la facilidad con la que se puede corregir un programa si se encuentra un error
Se utilizan medidas indirectas como:

Tiempo Medio de cambio (TMC)

Es decir, el tiempo que se tarda en:

- Analizar una petición
- Diseñar un modificación
- Implementar el cambio
- Probar y realizar el cambio.

Integridad

Mide la capacidad del software para resistir ataques.

Se debe tener en cuanta los siguientes atributos:

- Amenaza: Es la probabilidad de que un ataque ocurra en un tiempo determinado.
- Seguridad: Es la probabilidad de que se pueda repeler el ataque de un tipo determinado.

Se define como:

$$\textbf{Integridad} = \Sigma\ [(1\text{-amenaza}) \times (1\text{-seguridad})]$$

Facilidad de uso

Mide la "amigabilidad " del software con el usuario final.

Se mide en función de:

- Habilidad intelectual o física para aprender el sistema.
- El tiempo requerido para hacer uso eficiente del sistema.
- Aumento de la productividad.
- Valoración subjetiva de la disposición de los usuarios hacia el sistema.

2.5.2 Eficacia de la eliminación de defectos

La eficacia de la eliminación de defectos (EED), es una métrica que permite medir la habilidad de filtrar las actividades de la garantía de calidad y de control, ya que es aplicable a todas las actividades del marco de trabajo del proceso.

Se define de la siguiente forma:

$$EED = E / (E + D)$$

E	Número de errores encontrados antes de la entrega del software
D	Número de defectos encontrados después de la entrega

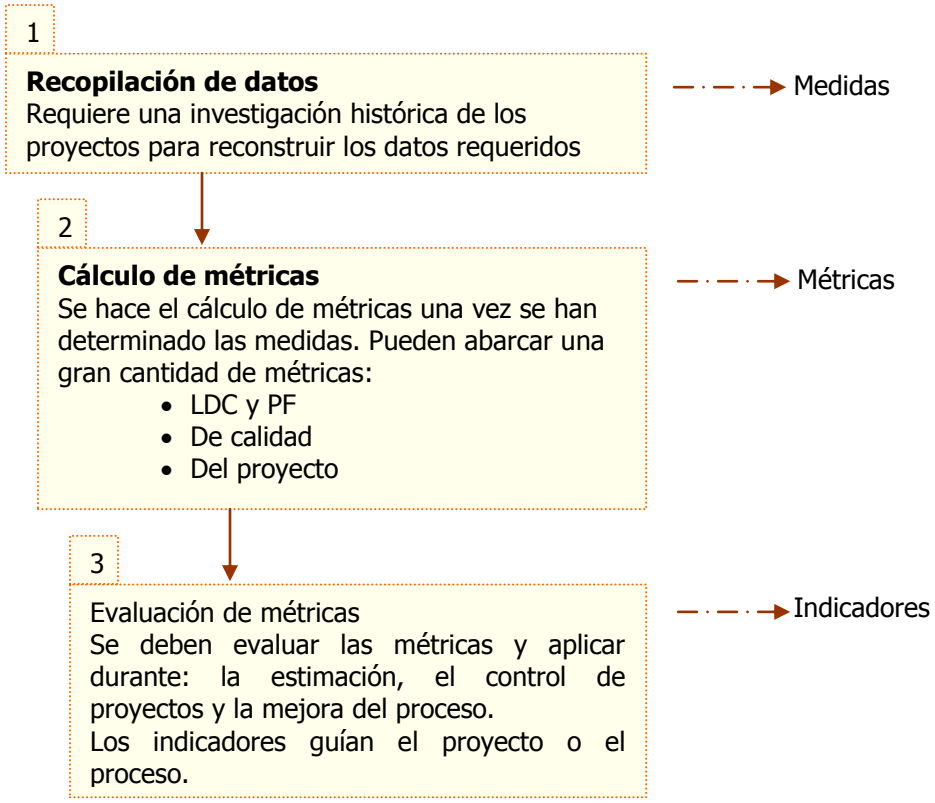
El valor ideal de EED es 1. ⟶ No se han encontrado defectos en el software.

2.5.3 Integración de las métricas dentro del proceso de Ingeniería del Software

Estableciendo una línea base de métricas se obtienen beneficios a nivel de:

- Proceso
- Proyecto
- Producto

Esta línea base, comprende los siguientes pasos:


1
Recopilación de datos
Requiere una investigación histórica de los proyectos para reconstruir los datos requeridos
Medidas
2
Cálculo de métricas
Se hace el cálculo de métricas una vez se han determinado las medidas. Pueden abarcar una gran cantidad de métricas:
LDC y PF
De calidad
Del proyecto
Métricas
3
Evaluación de métricas
Se deben evaluar las métricas y aplicar durante: la estimación, el control de proyectos y la mejora del proceso.
Los indicadores guían el proyecto o el proceso.
Indicadores

ACTIVIDADES COMPLEMENTARIAS

1. Describa, con sus propias palabras, la diferencia entre métricas del proceso y del proyecto.

2. Elabore un ensayo argumentativo, que de respuesta a la pregunta ¿Por qué renecesita medir? ¿Por qué son importantes las métricas dentro de la ingeniería de Software?

3. Sugiera tres medidas, tres métricas y los indicadores que se podrían utilizar para evaluar un automóvil.

4. Indague con algún desarrollador de software o un equipo de desarrollo de software qué medidas, métricas e indicadores utilizan o tienen implementadas. Elabore un cuadro sinóptico.

INVESTIGACIÓN

1. El Instituto de Ingeniería de Software (The Carnegie Mellon Software Engineering Institute -SEI) ha desarrollado una guía para establecer un programa de medición de software dirigido hacia objetivos. Investigue y elabore un documento sobre esta guía.

EJERCICIOS

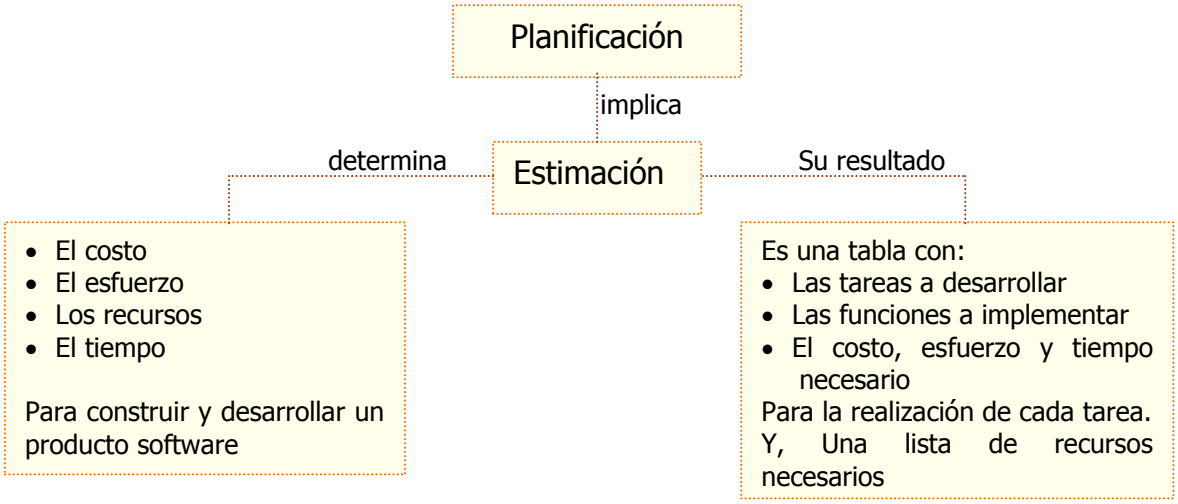
1. Calcule el Punto de Función de un proyecto con las siguientes características del dominio de información:

Número de entrada de usuario	32
Número de salida de usuario	60
Número de peticiones de usuario	24
Número de archivos	8
Número de interfaces externos	2

Asuma que todos los valores de ajuste de complejidad están en la media.

3. PLANIFICACIÓN DE PROYECTOS DE SOFTWARE

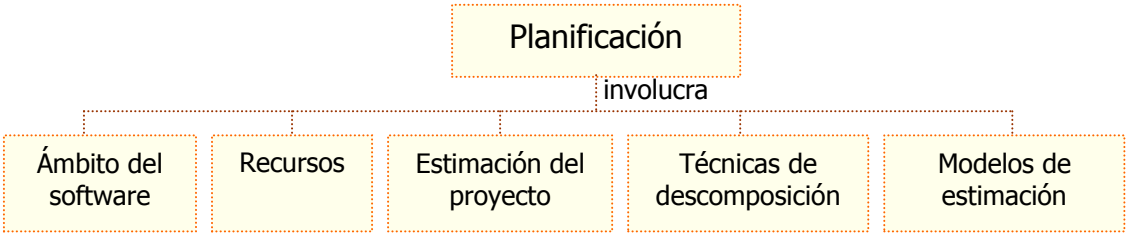
La Planificación del proyecto es el conjunto de actividades con las cuales comienza la gestión de proyectos software.



La estimación se inicia con una descripción del ámbito del producto. El problema se descompone en un conjunto de problemas de menor tamaño y cada uno de éstos se estima guiándose con datos históricos y con la experiencia.

El objetivo primordial es hacer estimaciones razonables de recursos, costos y una planificación temporal.

Las estimaciones involucran un periodo de tiempo y deben ser actualizadas a medida que avanza el proyecto.



3.1 Ámbito del Software y Recursos

3.1.1. Ámbito del Software

Describe el control y los datos a procesar, la función, el rendimiento, las restricciones, las interfaces y la fiabilidad. Se evalúan las funciones descritas en la declaración del ámbito, y en algunos casos se refinan para dar mas detalles antes del comienzo de la estimación.

Ámbito del software

comprende

Recolección de la información

Su objetivo es acercar al desarrollador y al cliente para establecer una comunicación, para lograr esto, se utiliza una técnica muy común que es una reunión o una entrevista preliminar.

Esta reunión o entrevista debe involucrar los siguientes tipos de preguntas:

1. Preguntas de contexto libre: se centran en el cliente, en los objetivos globales y en los beneficios. Estas preguntas deben llevar a un entendimiento básico del problema, las personas interesadas en la solución y la solución que se desea.

2. Metacuestiones: estas preguntas se centran en la efectividad de la reunión, involucra preguntas para determinar si la persona es la apropiada para responder a las preguntas, si son relevantes las preguntas para el problema en estudio, si las respuestas son oficiales, si existe algo que se debería preguntar.

También existe otra técnica que permite la creación de un equipo compuesto por los clientes y los desarrolladores para identificar el problema, proponer elementos de solución, establecer enfoques y especificar un conjunto preliminar de requisitos denominada TFEA (Facilitated application specification techniques) – Técnica para facilitar las especificaciones de la aplicación.

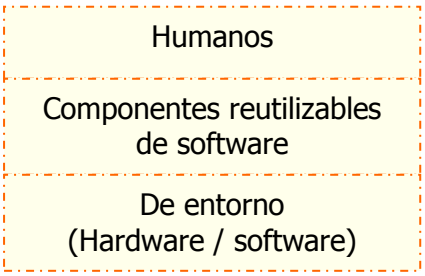
Viabilidad

Se centra en preguntarse:
¿Se puede construir el software de acuerdo al ámbito definido?
¿Es factible el proyecto?
La factibilidad del software tiene 4 dimensiones: Tecnología, financiación, Tiempo y Recursos. Tanto el equipo de desarrollo y las demás personas involucradas en el software deben determinar si puede ser construido dentro de las dimensiones especificadas.

3.1.2 Recursos

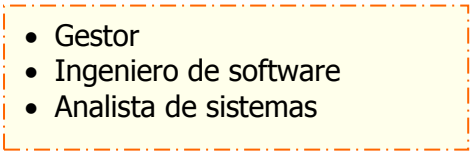
Comprende la estimación de los recursos necesarios para emprender el desarrollo del software.

Los recursos de desarrollo son:



Recurso humano

Se debe establecer el perfil y las habilidades que se necesitan del personal que se necesita para llevar a cabo el desarrollo del proyecto. Hay que especificar tanto la posición dentro de la organización como la especialidad.



El número de personas requerido para un proyecto de software se determina después de hacer una estimación del esfuerzo de desarrollo.

Recursos de software reutilizable

Se destaca la reutilización, esto es, la creación y la reutilización de bloques de construcción de software.

Se establecen 4 categorías de recursos de software que se deben tener en cuenta a medida que se avanza con la planificación:

- Componentes ya desarrollados: componentes que ya han sido validados totalmente se pueden utilizar e implementar en el desarrollo del proyecto actual.

- Componentes ya experimentados: se puede utilizar Especificaciones, diseños, código o datos de prueba existentes que ya han sido desarrollados para proyectos anteriores.

- Componentes con experiencia parcial: se puede utilizar Especificaciones, diseños, código o datos de prueba existentes que ya han sido desarrollados para proyectos anteriores y que requieren una modificación sustancial.

- Componentes nuevos: componentes que el equipo de software requiere construir específicamente para el proyecto.

Recursos de entorno

El entorno es donde se apoya el proyecto de software.

Comprende ------► Hardware y Software

Comprende el conjunto de herramientas requeridas para producir o desarrollar el producto software y se debe verificar que estos recursos estén disponibles.

3.2 Estimación del proyecto de software y técnicas de descomposición

Para realizar estimaciones seguras de costos y esfuerzos, se pueden tener las siguientes opciones:

1. Dejar la estimación para cuando el proyecto este más adelantado.
2. Basar las estimaciones en proyectos similares ya terminados.
3. Usar técnicas de descomposición que permita generar las estimaciones de costos y de esfuerzo del proyecto.
4. Utilizar modelos empíricos para la estimación del costo y esfuerzo del software.

La utilización de técnicas de descomposición y de modelos empíricos, permiten descomponer el proyecto en funciones principales y en tareas lo que implica que se pueda realizar una estimación del costo y del esfuerzo del proyecto de forma escalonada.

Antes de de realizar la estimación del proyecto se debe generar una estimación del tamaño del software a construir.

3.2.1 Tamaño del software

Dentro de la planificación de proyectos, el tamaño se refiere a una producción cuantificable del proyecto de software.

- El tamaño se mide en LDC, si se utiliza un enfoque directo
- El tamaño se representa como PF, si se utiliza un enfoque indirecto.

Se tienen 4 enfoques referentes al tamaño:

1

Tamaño en lógica difusa

Utiliza las técnicas aproximadas de razonamiento. Para aplicar este enfoque se debe:

- Identificar el tipo de aplicación
- Establecer su magnitud en una escala cuantitativa
- Refinar la magnitud dentro del rango original

¿Qué es Lógica Difusa?
Un tipo de lógica que reconoce más que simples valores verdaderos y falsos. Con lógica difusa, las proposiciones pueden ser representadas con grados de veracidad o falsedad. Por ejemplo, la sentencia "hoy es un día soleado", puede ser 100% verdad si no hay nubes, 80% verdad si hay pocas nubes, 50% verdad si existe neblina y 0% si llueve todo el día.

2

Tamaño de componentes estándar

El software esta compuesto por un número de componentes estándar (subsistemas, módulos, pantallas, informes, etc.) que son genéricos para un área en particular
Se debe:

- Estimar el número de incidencias de cada uno de los componentes
- Utilizar los datos de proyectos históricos para determinar el tamaño de entrega por componente.

Por ejemplo:
Para un sistema de información se estima que se requiere generar 15 informes. Los datos históricos indican que por informe se requieren 827 líneas de programación. Esto permite que se estime que se requieren 12405 LDC para el componente de informes.

3

Tamaño del cambio

Este enfoque se utiliza cuando en un proyecto se utiliza software existente y que se debe modificar de alguna manera como parte del proyecto.
Se debe estimar el número y tipo de modificaciones que se deben llevar a cabo.
Para estimar el tamaño del cambio, se utiliza una proporción de esfuerzo para cada tipo de cambio.

3.2.2 Estimación basada en el problema

- Puede usarse LOC o PF para hacer estimaciones.
- Si se utiliza LOC, la descomposición es esencial y a menudo debe ser a detalle.
- Si se utiliza PF, en vez de centrar la descomposición en la función, se calcula el PF, estimando de alguna forma, cada uno de los valores.
- En ambos casos, mediante datos históricos o la intuición, se estiman valores optimista (*O*), medio (*M*) y pesimista (*P*) para cada función o contador, y se calcula el valor esperado (*E*) con la siguiente fórmula:

$$E = (O + 4 * M + P) / 6$$

3.2.3 Estimación basada en el proceso

Esta técnica permite, descomponer el proceso en un conjunto relativamente pequeño de actividades o tareas, y en el esfuerzo requerido para llevar a cabo la estimación de cada tarea.

Esta estimación comprende:

1. Delineación de las funciones del software obtenidas a partir del ámbito del proyecto.
2. Se mezclan las funciones del problema y las actividades del proceso.
3. Se calculan los costos y el esfuerzo de cada función y la actividad del proceso de software.

3.3 Modelos empíricos de estimación

- Utilizan fórmulas derivadas empíricamente de una muestra limitada de proyectos para predecir el esfuerzo en función de LOC o PF.
- La estimación de los valores de LOC y PF se realizan utilizando las técnicas de descomposición.
- Los valores resultantes se aplican a la fórmula del modelo que se haya escogido para obtener el esfuerzo en hombres-mes.
- Precisamente por venir de una muestra limitada, no son adecuados para toda clase de software ni para todo medio ambiente de desarrollo.

Modelo COCOMO

Creado por Barry Boehm en 1981. Su nombre significa **CO**nstructive **CO**st **MO**del (Modelo constructivo de costo) y se puede dividir en tres modelos.

Modelos

COCOMO básico. Calcula el esfuerzo y el costo del desarrollo en función del tamaño del programa estimado en LOC.

COCOMO intermedio. Calcula el esfuerzo del desarrollo en función del tamaño del programa y un conjunto de conductores de costo que incluyen la evaluación subjetiva del producto, del hardware, del personal y de los atributos del proyecto.

COCOMO detallado. Incorpora las características de la versión intermedia y lleva a cabo una evaluación del impacto de los conductores de costo en cada fase (análisis, desarrollo, etc.) del proceso.

Los modelos COCOMO están definidos para tres tipos de proyectos de software:

- Orgánicos.
  - Proyectos pequeños y sencillos.
  - Equipos pequeños con experiencia en la aplicación.
  - Requisitos poco rígidos.

- Semiacoplados.
  - Proyectos de tamaño y complejidad intermedia.
  - Equipos con variado niveles de experiencia.
  - Requisitos poco o medio rígidos.

- Empotrados.
  - Proyectos que deben ser desarrollados con un conjunto de requisitos (hardware y software) muy restringidos.

COCOMO básico

Las ecuaciones del modelo COCOMO básico son de la forma:

$$E = a * KLOC^b$$
$$D = c * E^d$$

Donde:

E	es el esfuerzo aplicado en hombre-mes
D	es el tiempo de desarrollo en meses
KLOC	es el número de miles de líneas de código estimado para el proyecto

Los coeficientes *a* y *c* y los exponentes *b* y *d* se obtienen de la siguiente tabla:

Tipo de proyecto	***a***	***b***	***C***	***d***
Orgánico	2.4	1.05	2.5	0.38
Semiacoplado	3.0	1.12	2.5	0.35
Empotrado	3.6	1.20	2.5	0.32

Aplicando el modelo COCOMO básico al ejemplo anterior y usando un tipo de proyecto orgánico obtenemos una estimación para el esfuerzo:

$$E = 2.4 * KLOC^{1.05}$$
$$= 2.4 * 7.4^{1.05}$$
$$= 20 \text{ hombre-mes}$$

Para calcular la duración del proyecto usamos la estimación de esfuerzo:

$$D = 2.5 * E^{0.38}$$
$$= 2.5 * 20^{0.38}$$
$$= 8 \text{ meses}$$

El valor de la duración del proyecto permite al planificador recomendar un número de personas N para el proyecto.

$$N = E / D$$
$$= 20 / 8$$
$$= 3 \text{ personas}$$

El planificador puede decidir emplear sólo una o dos personas y ampliar por tanto la duración del proyecto.

COCOMO intermedio

En el COCOMO intermedio, la ecuación para calcular el tiempo de desarrollo es la misma que la del COCOMO básico. La ecuación para calcular el esfuerzo es:

$$E = a * KLOC^{b} * EAF$$

Donde,

E	es el esfuerzo en hombre-mes
KLOC	es el número estimado de miles de líneas de código

El coeficiente *a* y el exponente *b* están dados por la tabla:

Tipo de proyecto	***a***	***b***
Orgánico	3.2	1.05
Semiacoplado	3.0	1.12
Empotrado	2.8	1.20

Y

EAF	Es un factor de ajuste del esfuerzo que se calcula valorando en una escala de *muy bajo*, *bajo*, *nominal*, *alto* y *muy alto* cada uno de los siguientes 15 atributos, agrupados en 4 categorías. Así: **Atributos del producto.** Son restricciones y requerimientos del proyecto que va a ser desarrollado. • **Confiabilidad requerida*** • Tamaño de la base de datos • Complejidad del producto **Atributos de computadora.** Son limitaciones puestas por el hardware y el sistema operativo donde el proyecto va a ejecutarse. ○ **Restricciones de tiempo de ejecución*** ○ **Restricciones de memoria principal*** ○ Volatilidad de la máquina virtual. ○ Tiempo de respuesta de la computadora.

EAF	**Atributos de personal.** Nivel de habilidades que tiene el personal. Son habilidades profesionales generales, habilidad de programación, experiencia con el medio ambiente de desarrollo y familiaridad con el dominio del proyecto. ○ Capacidad del analista. ○ **Experiencia en aplicaciones*** ○ Capacidad del programador. ○ Experiencia con la máquina virtual. ○ Experiencia con el lenguaje de programación. **Atributos del proyecto.** Restricciones y condiciones bajo las cuales el proyecto se desarrolla. ○ Prácticas modernas de programación ○ **Uso de herramientas de software*** ○ Calendario de desarrollo requerido.

A cada atributo se le asigna un número real de acuerdo a la tabla siguiente:

Escala	Número
Muy bajo	0.75
Bajo	0.88
Nominal	1
Alto	1.15
Muy alto	1.40

El número indica el grado con el que cada factor puede influenciar la productividad. Un valor menor que 1 indica que el factor puede decrementar el calendario y el esfuerzo. Un valor mayor que 1 denota un factor que extiende el calendario y el esfuerzo. Finalmente, un valor igual a 1 no extiende ni decrementa el calendario y el esfuerzo (esta clase de factor se llama nominal).
Para obtener el EAF se multiplican cada uno de los 15 factores.
Se puede simplificar el cálculo del EAF porque hay una tendencia a considerar los atributos marcados en (*****), como los más relevantes y que deberían ser tomados en cuenta.

La ecuación del software

- Propuesta por Putnam y Myers en 1992.
- Asume una distribución específica del esfuerzo a lo largo de la vida de un proyecto.
- Se obtuvo a partir de datos de productividad de unos 4,000 proyectos.

Es de la forma:

$$E = (LOC * B^{0.333} / P)^3 * (1 / t^4)$$

Donde,

E	Esfuerzo en hombres-año.
t	Duración del proyecto en años.
B	Factor especial de destrezas. Para programas pequeños *B* vale 0.16, para programas intermedios vale 0.28, para programas mayores vale 0.39.
P	Parámetro de productividad, para un software de tiempo real, *P* vale 2,000, para comunicaciones vale 10,000, para software científico vale 12,000 y para aplicaciones comerciales de sistemas vale 28,000.

- Para simplificar el proceso de estimación se sugiere un conjunto de ecuaciones obtenidas de la ecuación del software.
- Un tiempo mínimo de desarrollo de software se define como:

$$t_{min} = 8.14 * (LOC / P)^{0.43} \text{ para } t_{min} > 6 \text{ meses.}$$

$E = 180 * B * t^3$ en hombres-mes para E >= 20 hombres-mes y t representado en años

Aplicando las ecuaciones al ejemplo anterior, obtenemos:

$$t_{min} = 8.14 * (7{,}400 / 12{,}000)^{0.43}$$
$$= 7 \text{ meses}$$
$$E = 180 * 0.28 * 0.75^3$$
$$= 21 \text{ hombres-mes}$$

3.4 Riesgo del software

El análisis y la gestión del riesgo son una serie de pasos que ayudan al equipo del software a comprender y a gestionar la incertidumbre.

Un riesgo es un problema potencial – puede ocurrir o no -.

Por tal razón es importante:

- Identificarlo
- Evaluar su probabilidad de aparición,
- Estimar su impacto, y ,
- Establecer un plan de contingencia por si ocurre el problema.

Los pasos que se deben seguir son:

1. Identificar el riesgo. Reconocimiento de que algo puede ir mal.
2. Cada riesgo se analiza para determinar la probabilidad de que pueda ocurrir y el daño que puede causar si ocurre.
3. Se priorizan los riesgos, en función de la probabilidad y del impacto.
3. Se desarrolla un plan para gestionar aquellos riesgos con gran probabilidad e impacto.

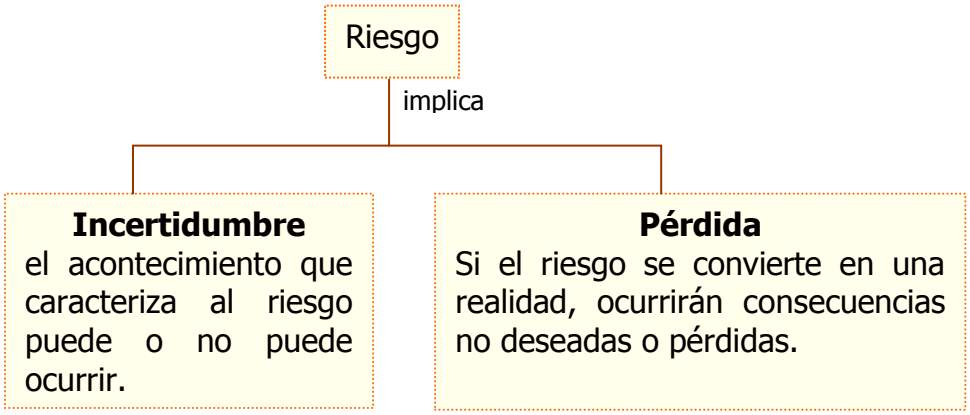
3.4.1. Estrategias de riesgo

Se tienen dos estrategias:

Reactiva
Supervisa el proyecto en previsión de posibles riesgos.
- Evaluar las consecuencias del riesgo cuando este ya se ha producido (ya no es un riesgo)
- Actuar en consecuencia

Proactiva
Identifica los riesgos potenciales.
- Evaluación previa y sistemática de riesgos
- Evaluación de consecuencias

Se establece un plan de contingencia para controlar el riesgo.



3.4.2 Categorías de riesgos

Riesgos del proyecto
Pueden amenazar al plan del proyecto, porque puede retrazar el proyecto y aumentar los costos.
Identifican problemas de:

- Presupuesto
- Personal
- Recursos
- Cliente
- Requisitos

Riesgos técnicos
Amenazan la calidad del software y la planificación temporal del proyecto. La implementación puede llegar a ser difícil o imposible.
Identifican problemas de:

- Diseño
- Implementación
- De interfaz
- Verificación
- Mantenimiento

Riesgos del negocio
Amenazan la viabilidad del software a construir. Se pone en riesgo el proyecto y el producto.
Identifican problemas de:

- De mercado
- De estrategia
- De ventas
- De gestión
- De presupuesto

3.4.3 Identificación del riesgo

Existen dos tipos de riesgo:

Riesgos genéricos	**Riesgos específicos**
Representan una amenaza potencial para todos los proyectos de software.	Implican un conocimiento profundo del proyecto (Entorno, Tecnología, Personal)

Lista de comprobación de elementos de riesgo

Es un método que permite identificar riesgos, por medio de las siguientes categorías:

Relacionados con el tamaño del producto	Se asocian los riesgos con el tamaño del software a construir o desarrollar. • Tamaño estimado del proyecto (LOC/PF) • Confianza en la estimación • Numero de programas, archivos y transacciones • Tamaño relativo al resto de proyectos • Tamaño de la base de datos • Número de usuarios • Número de cambios de requerimientos previstos antes y después de la entrega • Cantidad de software reutilizado
Con el impacto en la organización	Asociados con las limitaciones impuestas por la gestión. • Efecto del producto en la cifra de ventas • Visibilidad desde la dirección de la organización • Fecha límite de entrega razonable • Número de clientes que usarán el producto • Numero de productos con los que deberá interaccionar • Sofisticación del usuario final • Cantidad y calidad de la documentación a entregar al cliente • Límites legales y gubernamentales • Costos asociados al retraso en la entrega • Costos asociados a errores en el producto

Con el tipo de cliente	Asociados con la comunicación del cliente. • Hay experiencias anteriores con dicho cliente • Tiene una idea clara de lo que precisa • Está dispuesto a dedicar tiempo en la especificación formal de requerimientos • Está dispuesto a relacionarse de forma ágil con el equipo de desarrollo • Está dispuesto a participar en la revisiones • Es un usuario experto • Dejará trabajar al equipo de desarrollo sin dar consejos de *experto informático* • Entiende el ciclo de vida de una aplicación
Con la definición del proceso de producción	Asociados al proceso y seguimiento del software. • Hay una política clara de normalización y seguimiento de una metodología • Existe una metodología escrita para el proyecto • Se ha utilizado en otros proyectos • Están los gestores y desarrolladores formados • Conoce todo el mundo los estándares • Existen plantillas y modelos para todos los documentos resultado del proceso • Se aplican revisiones técnicas de la especificación de requerimientos diseño y codificación • Se aplican revisiones técnicas de los procedimientos de revisión y prueba • Se documentan los resultados de las revisiones técnicas • Hay algún mecanismos para asegurar que un proceso de desarrollo sigue los estándares • Se realiza gestión de la configuración • Hay mecanismos para controlar los cambios en los requerimientos que tienen impacto en el software • Se documenta suficientemente cada subcontrato • Se ha habilitado y se siguen mecanismos de seguimiento y evaluación técnica de cada subcontrato. • Se dispone de técnicas de especificación de aplicaciones para facilitar la comunicación con el cliente. • Se usan métodos específicos para análisis de software • Se utiliza un método específico para el diseño arquitectónico y de datos • Está el 90% del código en lenguajes de alto nivel

	• Hay estándares de documentación de código • Se usan métodos específicos para el diseño de pruebas • Se utilizan herramientas para llevar a cabo la planificación y control

Con el entorno de desarrollo	Asociados a las herramientas que se van a utilizar en el desarrollo del software • Hay herramientas de gestión de proyectos • Hay herramientas de gestión del proceso de desarrollo • Hay herramientas de análisis y diseño • Hay generadores de código apropiados para la aplicación • Hay herramientas de prueba apropiadas • Hay herramientas de gestión de configuración apropiadas • Se hace uso de una base de datos o repositorio centralizado • Están todas las herramientas de desarrollo integradas • Se ha proporcionado formación a todos los miembros del equipo de desarrollo • Hay expertos a los cuales solicitar ayuda acerca de las herramientas • Hay ayuda en línea y documentación disponible
Con la tecnología	Asociados con la tecnología a utilizar y la complejidad del sistema • Se trata de una tecnología nueva en la organización • Se requieren nuevos algoritmos o tecnología de I/O • Se debe interactuar con hardware nuevo • Se debe interactuar con software que no ha sido probado • Se debe interactuar con un B.D. cuya funcionalidad y rendimiento no ha sido probada • Es requerido un interface de usuario especializado • Se necesitan componentes de programa radicalmente diferentes a los hasta ahora desarrollados • Se deben utilizar métodos nuevos de análisis, diseño o pruebas • Se deben utilizar métodos de desarrollo no habituales, tales como métodos formales, Inteligencia Artificial o redes neuronales • Se aplican requisitos de rendimiento especialmente estrictos

	• Existen dudas de que el proyecto sea realizable
Con la experiencia técnica	Asociados a la experiencia de los ingenieros de desarrollo de software. • Es el mejor personal disponible • Tienen los miembros las técnicas apropiadas • Hay suficiente gente disponible • Está el personal comprometido en toda la duración del proyecto • Habrá parte del personal dedicado solamente en parte al proyecto • Tiene el personal las expectativas correctas del trabajo • Tiene el personal la necesaria formación • Puede la rotación del personal perjudicar el proceso de desarrollo

3.4.4 Proyección del riesgo

La estimación del riesgo mide el riesgo de dos maneras:

- La probabilidad de que el riesgo sea real
- Las consecuencias de los problemas asociados con el riesgo, si ocurriera

Se realizan cuatro actividades de proyección del riesgo:

1. Establecer una escala que refleje la probabilidad percibida del riesgo
2. Definir las consecuencias del riesgo
3. Estimar el impacto del riesgo en el proyecto y en el producto
4. Apuntar la exactitud general de la proyección del riesgo de manera que no haya confusiones.

Uso de Tablas de Riesgo

Por medio del uso de la siguiente tabla se facilita una proyección del riesgo.

Riesgos	Categoría	Probabilidad	Impacto
Mayor número de usuarios previstos	TP	30%	3

1. En la columna **Riesgo**, se registran todos los riesgos
2. En la columna **Categoría**, cada riesgo se categoriza así:

- Tamaño del producto (TP)
- Impacto en la organización (IO)
- Tipo de cliente (TC)
- Proceso de producción (PP)
- Entorno de desarrollo (ED)
- Tecnología (T)
- Experiencia técnica (ET)

Se pueden utilizar las iniciales que se encuentran entre paréntesis o puede asignar unas específicas.

3. En la columna **Probabilidad**, se registra la probabilidad de aparición de cada riesgo.
4. En la columna **Impacto**, Se valora y se registra el impacto de cada riesgo así:

1	Catastrófico
2	Crítico
3	Marginal
4	Despreciable

Por último la tabla es ordenada por probabilidad y por impacto. Aquellos riesgos que presentan alta probabilidad y alto impacto pasan al inicio de la tabla y los que presentan baja probabilidad e impacto pasan al final de la tabla.

Una vez la tabla ha sido ordenada, el encargado del proyecto debe trazar una línea de corte donde los riesgos que se encuentren por encima de ésta línea se les prestara una mayor atención.

Evaluación del impacto del riesgo

La exposición al riesgo está dada por la ecuación:

$$ER = P \times C$$

Donde:

P	Es la probabilidad de que ocurra un riesgo
C	Costo del proyecto si el riesgo ocurre

Esta ecuación se aplica para calcular la exposición al riesgo de cada riesgo descrito en la tabla de riesgos. Estos cálculos permiten ajustar los costos finales del proyecto.

Evaluación del riesgo

Se establece un punto en el que se decide cuándo desistir y finalizar el proyecto. Para esto se debe:

- Definir un punto de referencia
- Marcar la relación entre cada factor de riesgo enumerado y el punto de referencia
- Definir el área de incertidumbre, donde será tan válido continuar como interrumpir el trabajo
- Predecir cómo la combinación de riesgos afectará a los niveles de referencia

3.4.5 Reducción, supervisión y gestión del riesgo

El objetivo que debe tener un equipo de software es evitar y tratar un riesgo. Para esto, es importante que se desarrolle un plan de reducción del riesgo.

Este plan de reducción del riesgo involucra para cada riesgo una serie de pasos y acciones que debe tomar e implementar el equipo de desarrollo del software.

El plan RSGR

Se puede incluir una estrategia de gestión de riesgo en el plan del proyecto de software o se podrían organizar los pasos de gestión del riesgo en un plan diferente de reducción, supervisión y gestión del riesgo (Plan RSGR). Todos los documentos del plan RSGR se llevan a cabo como parte del análisis de riesgo y son empleados por el jefe del proyecto como parte del Plan del Proyecto general.

A continuación se expone un esquema del Plan RSGR:

I. Introducción
    1. Alcance y propósito del documento
    2. Visión general de los riesgos principales
    3. Responsabilidades
        a. Gestión
        b. Personal técnico
II. Tabla de riesgo del proyecto.
    1. Descripción de todos los riesgos por encima de la línea de corte
    2. Factores que influyen en la probabilidad e impacto
III. Reducción, supervisión y gestión del riesgo
    n. Riesgo # n
    a. Reducción
        i.Estrategia general.
        ii. Pasos específicos.
    b. Supervisión
        i. Factores a supervisar
        ii. Enfoque de supervisión
    c. Gestión
        i. Plan de contingencia
        ii. Consideraciones especiales.
IV. Planificación temporal de revisión del Plan RSGR
V. Resumen

Una vez que se ha desarrollado el plan RSGR y el proyecto ha comenzado, empiezan los procedimientos de reducción y supervisión del riesgo.

La reducción del riesgo es una actividad para evitar problemas.

La supervisión del riesgo es una actividad de seguimiento del proyecto con tres objetivos principales:

1. Valorar cuando un riesgo previsto ocurre de hecho.

2. Asegurarse de que los procedimientos para evitar el riesgo definidos para el riesgo en cuestión se están aplicando apropiadamente.

3. Recoger información que pueda emplearse en el futuro para analizar riesgos.

La supervisión de riesgos también intentar determinar el "origen" (qué riesgos ocasionaron tal problema) a lo largo de todo el proyecto.

3.5 Planificación temporal

La planificación temporal de un proyecto es una actividad que evoluciona con el tiempo y que permite identificar, definir y programar las actividades específicas que se requieren para realizar una actividad.

La planificación temporal permite:

- Definir todas las tareas
- Definir las tareas críticas
- Identificar el camino crítico
- Realizar seguimiento a tareas -> Detectar retraso

3.5.1 Principios básicos de la planificación temporal

Compartimentar	Tareas y actividades manejables
Interdependencia	Las actividades pueden ser: • Secuenciales • Paralelas • Independientes • Orden de ejecución
Asignación de tiempo	A cada tarea se debe asignar: • Nº unidades de tiempo • Fecha inicio y fecha fin
Validación de esfuerzo	Asignar: Esfuerzo ← Nº personas actual
Definir responsabilidades	Asignar: Tarea ← Miembro equipo
Definir resultados	Cada tarea debe tener un resultado o producto definido.
Hitos	Cada tarea ← Hito

3.5.2 Herramientas De Planificación Temporal

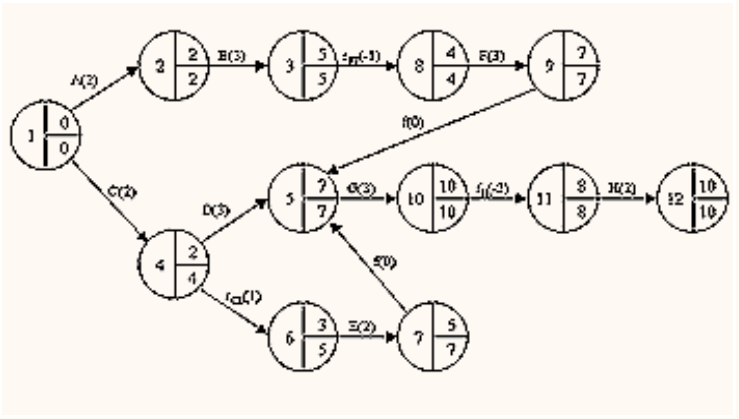
1. PERT (Program Evaluation and Review Technique)

El diagrama PERT es una representación gráfica de las relaciones entre las tareas del proyecto que permite calcular los tiempos del proyecto de forma sencilla.

Características

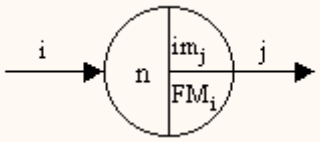
- Es un grafo, o sea, un conjunto de puntos (nodos) unidos por flechas.
- Representa las relaciones entre las tareas del proyecto, no su distribución temporal.
- Las flechas del grafo corresponden a las tareas del proyecto.
- Los nodos del grafo, representado por círculos o rectángulos, corresponden a instantes del proyecto. Cada nodo puede representar hasta dos instantes distintos, el inicio mínimo de las tareas que parten del nodo y el final máximo de las tareas que llegan al mismo.
- Es una herramienta de cálculo, y una representación visual de las dependencias entre las tareas del proyecto.

Tarea	Predec.	Duración
A	-	2
B	A	3
C	-	2
D	C	3
E	D_{II+1}	2
F	B_{FI-1}	3
G	D, E, F	3
H	G_{FF}	2

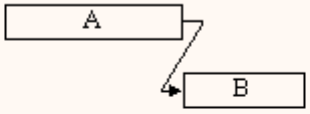


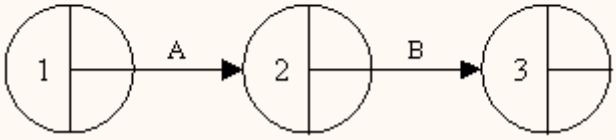
Para construir un diagrama PERT se deben tener en cuenta las siguientes reglas

- Los nodos representan instantes del proyecto. Cada nodo representa el inicio mínimo (im) de las tareas que tienen origen en dicho nodo y el final máximo (FM) de las tareas que llegan al mismo.

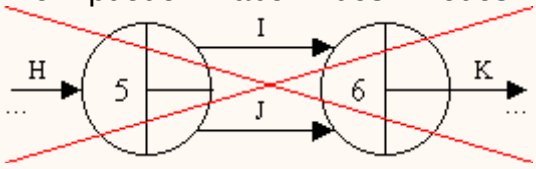


- Sólo puede haber un nodo inicial y un nodo final. O sea, sólo puede haber un nodo al que no llegue ninguna flecha (nodo inicial) y sólo puede haber un nodo del que no salga ninguna flecha (nodo final).
- La numeración de los nodos es arbitraria, si bien se reservan el número menor (generalmente el 0 o el 1) para el nodo inicial y el mayor para el nodo final.
- Las flechas representan tareas y se dibujan de manera que representen las relaciones de dependencia entre las tareas. Los recorridos posibles a través del diagrama desde el nodo inicial al nodo final, siguiendo el sentido de las flechas, deben corresponder con las secuencias en que deben realizarse las distintas tareas, o sea, los caminos del proyecto.





- No puede haber dos nodos unidos por más de una flecha.



Se pueden introducir tareas ficticias, de duración 0, para evitar construcciones ilegales o representar dependencias entre tareas.

Con un diagrama PERT se obtiene un conocimiento preciso de la secuencia necesaria, o planificada para la ejecución de cada actividad y utilización de diagramas de red.

Se trata de un método muy **orientado al plazo de ejecución**, con poca consideración hacia al costo.

Se suponen tres duraciones para cada suceso:

- la optimista a,
- la pesimista b y
- la normal m;

Suponiendo una distribución beta, la duración más probable: t = (a + 4m + b) / 6.

Aplicación de las técnicas PERT:

- Determinar las actividades necesarias y cuando lo son.
- Buscar el plazo mínimo de ejecución del proyecto.
- Buscar las ligaduras temporales entre actividades del proyecto.
- Identificar las actividades críticas, es decir, aquellas cuyo retraso en la ejecución supone un retraso del proyecto completo.
- Identificar el camino crítico, que es aquel formado por la secuencia de actividades críticas del proyecto.
- Detectar y cuantificar las holguras de las actividades no críticas, es decir, el tiempo que pueden retrasarse (en su comienzo o finalización) sin que el proyecto se vea retrasado por ello.
- Si se está fuera de tiempo durante la ejecución del proyecto, señala las actividades que hay que forzar.
- Nos da un proyecto de coste mínimo.

2. CPM (Crítical Path Method) Método del camino Crítico

El camino crítico en un proyecto es la sucesión de actividades que dan lugar al máximo tiempo acumulativo. Determina el tiempo más corto que podemos tardar en hacer el proyecto si se dispone de todos los recursos necesarios. Es necesario conocer la duración de las actividades.

Este concepto es utilizado por dos métodos:

- Método del tiempo estimado (CPM) La duración de una actividad es la más probable de duración. Tiempo que se emplearía en condiciones normales (m). Situación determinista.
- Método del tiempo esperado (PERT) Determinación probabilística de los tiempos esperados (Te), en función de los siguientes tiempos:
  - Duración más corta (a)
  - Duración más larga (b)
  - Duración más probable (m) (el mismo que en CPM)
  - Duración esperada: Te = (a + 4m + b) / 6

Cálculo del camino crítico

1. Calcular Te ó m según el método empleado para cada actividad. Se coloca en el grafo encima o debajo de cada flecha.
2. Calcular las fechas "early" -fecha mínima de comienzo de la actividad, MIC del suceso anterior- y "last" -fecha mínima de comienzo de la actividad, MAC del suceso posterior- de las distintas actividades que configuran el proyecto. (Calcular el MIC y el MAC de todos los sucesos del proyecto).
3. Cálculo de las holguras.
4. Identificación del camino crítico.

Holguras

La holgura de una actividad es el margen suplementario de tiempo que tenemos para determinar esa actividad. Las actividades críticas no tienen holgura.

Holgura de un suceso "Hs":	Hs = MAC del suceso – MIC del suceso
Holgura total de una actividad "Ht":	Ht = MAC del s.p. – MIC del s.a. – duración tarea
Margen suplementario de tiempo de esa actividad sin que se altere el MIC de ninguna actividad crítica.	
Holgura libre de una"Hi":	HI = MIC del s.p. – MIC del s.a. – duración tarea
Margen suplementario de tiempo para esa actividad sin que se altere el MIC de cualquier actividad.	

Holgura independiente "Hi":	Hi = MIC del s.p. – MAC del s.a. – duración tarea
Margen suplementario de tiempo que existe en una actividad si las actividades precedentes terminaran lo más tarde posible, y las actividades posteriores empezaran lo antes posible.	

Actividades críticas

Una actividad es crítica cuando no se puede cambiar sus instantes de comienzo y finalización sin modificar la duración total del proyecto. La concatenación de actividades críticas es el camino crítico.

En una actividad crítica la fecha "early" coincide con la más tardía de comienzo, y la fecha más temprana de finalización coincide con la fecha "last" de la actividad. La holgura total es 0.

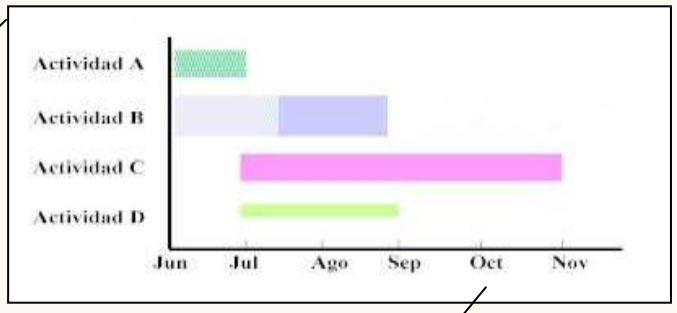
3. DIAGRAMA DE GANTT

Los cronogramas de barras o "gráficos de Gantt" fueron concebidos por el ingeniero norteamericano Henry L. Gantt.

Gantt resolvió el problema de la programación de actividades, es decir, su distribución conforme a un calendario, donde se podía visualizar el periodo de duración de cada actividad, sus fechas de iniciación y terminación e igualmente el tiempo total requerido para la ejecución de un trabajo. El instrumento que desarrolló permite también que se siga el curso de cada actividad, al proporcionar información del porcentaje ejecutado de cada una de ellas, así como el grado de adelanto o atraso con respecto al plazo previsto.

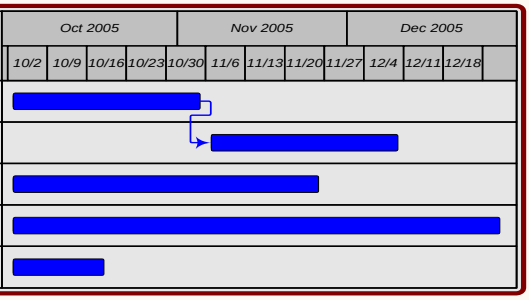
Este gráfico consiste en un sistema de coordenadas en que se indica:

En el eje Vertical: Las actividades que constituyen el trabajo a ejecutar. A cada actividad se hace corresponder una línea horizontal cuya longitud es proporcional a su duración en la cual la medición efectúa con relación a la escala definida en el eje horizontal.



En el eje Horizontal: un calendario, o escala de tiempo definido en términos de la unidad más adecuada al trabajo que se va a ejecutar: hora, día, semana, mes, etc.

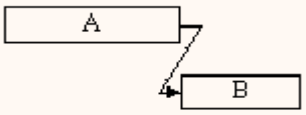
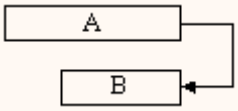
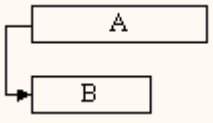
ID	Actividades	Inicio	Fin	Duracion
1	Actividad 1	03/10/2005	04/11/2005	5w
2	Actividad 2	07/11/2005	09/12/2005	5w
3	Actividad 3	03/10/2005	25/11/2005	8w
4	Actividad 4	03/10/2005	27/12/2005	12.40w
5	Actividad 5	03/10/2005	18/10/2005	2.40w



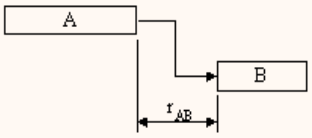
Símbolos Convencionales: Los símbolos básicos son los siguientes:

- Iniciación de una actividad.
- Término de una actividad
- Línea fina que conecta las dos "L" invertidas. Indica la duración prevista de la actividad.
- Línea gruesa. Indica la fracción ya realizada de la actividad, en términos de porcentaje. Debe trazarse debajo de la línea fina que representa el plazo previsto.
- Plazo durante el cual no puede realizarse la actividad. Corresponde al tiempo improductivo puede anotarse encima del símbolo utilizando una abreviatura.
- Indica la fecha en que se procedió a la última actualización del gráfico, es decir, en que se hizo la comparación entre las actividades previstas y las efectivamente realizadas.

En el proceso de dibujar un diagrama de Gantt se tendrán en cuenta las siguientes consideraciones siguientes:

Las dependencias fin-inicio se representan alineando el final del bloque de la tarea predecesora con el inicio del bloque de la tarea dependiente.	
Las dependencias final-final se representan alineando los finales de los bloques de las tareas predecesora y dependiente.	
Las dependencias inicio-inicio se representan alineando los inicios de los bloques de las tareas predecesora y dependiente.	

Los retardos se representan desplazando la tarea dependiente hacia la derecha en el caso de retardos positivos y hacia la izquierda en el caso de retardos negativos.



El diagrama de Gantt es un diagrama representativo, que permite visualizar fácilmente la distribución temporal del proyecto, pero es poco adecuado para la realización de cálculos.

ACTIVIDADES COMPLEMENTARIAS

1. Investigue sobre TFEA (Facilitated application specification techniques). Puede consultar en:

   http://www.rspa.com/checklists/risk.html

2. Investigue y profundice sobre el tema: Estimación del proyecto software. Elabore un ensayo.

3. Un tema de gran importancia en el cual se puede profundizar es:
   - Lógica difusa
   - Constructive Cost Model

4. Describa la diferencia entre “riesgos conocidos” y “riesgos predecibles”

5. Usted es el jefe de proyectos de una compañía de software. Se le ha pedido que dirija a un equipo que está desarrollando un software de un procesador de textos. Construya una tabla de riesgo para el proyecto.

6. Asuma que ha sido contratado por una Universidad para desarrollar un sistema de inscripción a cursos para un determinado programa. Defina un listado de tareas. Utilice las diferentes técnicas descritas en el capítulo para establecer una planificación temporal del proyecto.

BIBLIOGRAFIA

IMPRESA

BRAUDE. Ingeniería de software, una perspectiva orientada a objetos. México. 2003. Alfaomega grupo editor. S.A.

GRUEGGE, BERND y DUTOIT, Allen H. Ingeniería de software orientado a objetos. México. 2002. Pearson Educación.

HUMPHREY, Watts S. Introducción al proceso de software personal. Pearson Addison wesley. 2001.

MEYER, Bertrand. Construcción de software orientado a objetos. Segunda edición. Madrid. 1999. Prentice Hall.

NORRIS. Ingeniería de software explicada. Grupo Noriega editores de Colombia.

PIATTINI, Mario. VILLALBA, Jose y otros. Mantenimiento del software: modelos, técnicas y métodos para la gestión del cambio. Editorial Alfaomega-Rama.

PRESSMAN, Roger S. Ingeniería del Software. Un enfoque práctico. Quinta edición. España. 2002. Editorial McGraw Hill.

PFLEEGER, Shari Lawrence. Ingeniería de software, teoría y práctica. 1ª. Edición. Buenos Aires. Pearson educación. 2002

SOMMERVILLE, Ian. Ingeniería de software. 6ª. Edición. Pearson Addison Wesley. 2001

ELECTRÓNICA

http://www.rspa.com
http://www.pmi.org
http://www.4pm.com
http://www.projectmanagement.com
http://www.salud.gob.mx/unidades/dgces/gestion/page/05.html
www.ifpug.org
http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html
www.qsm.com
www.spr.com

UNIDAD 3.
CONTROL DE CALIDAD DEL SOFTWARE

INTRODUCCIÓN

La garantía de calidad del software es una actividad de protección que se aplica a cada paso del proceso de software y que comprende: procedimientos, métodos y herramientas, revisiones técnicas formales, técnicas y estrategias de prueba, procedimientos de garantía de ajustes y mecanismos de medida e información.

OBJETIVOS

GENERALES

- Estudiar las técnicas de gestión de calidad del software.
- Determinar las técnicas de prueba de software con el propósito de encontrar y corregir errores antes de entregar el programa al cliente.
- Definir las estrategias de prueba del software
- Determinar y analizar las métricas técnicas del software
- Determinar y aprender los beneficios de la reutilización del software.

ESPECIFICOS

- Determinar qué es la calidad del software
- Identificar los aspectos de gestión y las actividades específicas del proceso de calidad del software.
- Establecer la importancia de la garantí de calidad del software así como definir las estrategias para los planes de garantía de calidad del software.
- Definir los fundamentos de las pruebas del software.
- Determinar que son las pruebas de caja negra, blanca, de camino básico y de estructura de control.
- Identificar las pruebas de unidad, integración, validación y del sistema.
- Identificar las métricas del modelo de análisis, del modelo de diseño, del código fuente, para pruebas y del mantenimiento.
- Definir los fundamentos de la reutilización del software.
- Determinar las dificultades para la reutilización
- Determinar algunas sugerencias para establecer un enfoque de la reutilización.

1. GARANTIA DE CALIDAD DEL SOFTWARE

La garantía de calidad del software (SQA, Software Quality Assurance GCS, Gestión de calidad del software) es una actividad de protección que se aplica a lo largo de todo el proceso del software.

[2]Antes del siglo veinte, la garantía de calidad era responsabilidad única de la persona que construía el producto. La primera función de control y de garantía de calidad formal fue introducida por los laboratorios Bell en 1916 y se extendió rápidamente por todo el mundo de las manufacturas. Hoy en día, cada compañía tiene un mecanismo que asegura la calidad de sus productos de hecho, durante la pasada década, se han usado ampliamente como tácticas de mercado la declaración explícita de mensajes que ponían de manifiesto la calidad ofrecida por las compañías.

La historia de la garantía de calidad en el desarrollo de software ha sido paralela a la historia en la fabricación de hardware. Durante los primeros años de información (los 50 y los 60), la calidad era responsabilidad únicamente del programador.

Durante los años 70 se introdujeron estándares de garantía de calidad para el software en los contratos militares de desarrollo de software y se han extendido rápidamente en los desarrollos de software del mundo comercial.

La SQA forma parte de una función más amplia de garantía de calidad y engloba las siguientes actividades:

1. Un enfoque de gestión de calidad.
2. Métodos y herramientas
3. Revisiones técnicas formales
4. Documentación

La calidad del software es importante porque:

- Se reduce la repetición de actividades o tareas.
- Supone costos más bajos de desarrollo.
- Se mejora el proceso del software y por ende el producto.

1.1 Conceptos de calidad

[2] http://www.pcm.gob.pe/portal_ongei/publicaciones/cultura/Lib5042/cap20.htm

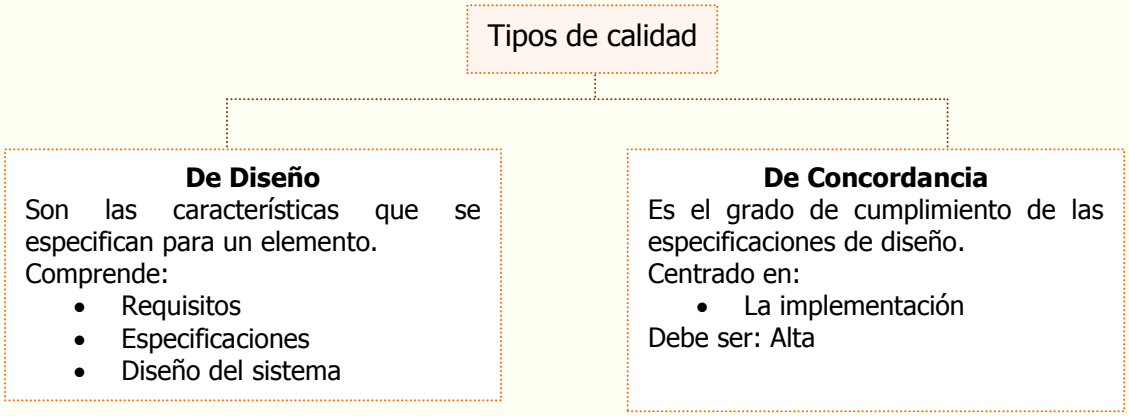
1. Calidad

Según ISO 9000: la calidad es el conjunto de características de una entidad que le confieren su aptitud para satisfacer las necesidades expresadas y las implícitas.

Según Roger Preesman: Calidad se define como “una característica o atributo de algo”.

Otras definiciones plantean:

- Es la medida en que las propiedades de un bien o servicio cumplen con los requisitos establecidos en la norma o especificaciones técnicas, así como con las exigencias del usuario de dicho bien o servicio en cuanto a su funcionalidad, durabilidad y costo.
- Aquellas características del producto que responden a las necesidades del cliente.
- El significado histórico de la palabra calidad es el de aptitud o adecuación al uso.
  Se dice que un producto o servicio es de calidad cuando satisface las necesidades y expectativas del cliente o usuario, en función de parámetros como:
    - Seguridad que el producto o servicio confieren al cliente.
    - Fiabilidad o capacidad que tiene el producto o servicio para cumplir las funciones especificadas, sin fallo y por un período determinado de tiempo.
    - Servicio o medida en que el fabricante y distribuidor responden en caso de fallo del producto o servicio.
- La Sociedad Americana para el Control de Calidad (A.S.Q.C.), define la calidad como el conjunto de características de un producto, proceso o servicio que le confieren su aptitud para satisfacer las necesidades del usuario o cliente.



2. Control de Calidad

A nivel general, se pueden tener las siguientes definiciones:

- El control de calidad comprende aquellas actividades realizadas en una empresa u organización para aplicar los principios de calidad en las actividades realizadas en la empresa.
- Es la actividad mediante la cual una empresa determina si el producto que elabora o el servicio que presta cumple o no, con las especificaciones contenidas en la Norma de calidad específica para tal producto o servicio.
- El control de calidad se ocupa de garantizar el logro de los objetivos de calidad del trabajo respecto a la realización del nivel de calidad previsto para la producción y sobre la reducción de los costos de la calidad.

Según ISO:

"Conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requerimientos relativos a la calidad del producto o servicio".

En Ingeniería de Software,

Control de Calidad es el conjunto de inspecciones, revisiones y pruebas utilizadas a lo largo del proceso del software para asegurar que cada producto cumple con los requisitos que le han sido asignados.

En el control de calidad:

1. Se deben definir todos los productos y las especificaciones mensurables en las que se puedan comparar los resultados de cada proceso.
2. Las actividades pueden ser automáticas, manuales o combinadas.
3. Existe un feedback (realimentación)

3. Garantía de Calidad

A nivel General:

- La garantía de la calidad o aseguramiento de calidad comprende todas aquellas actividades de una empresa u organismo para conseguir y demostrar la calidad en ésta. Estas actividades se dividen en tres apartados:
  - Evaluación de la calidad
  - Control de calidad
  - Correcciones internas

Según ISO:
Conjunto de acciones planificadas y sistemáticas necesarias para proporcionar la confianza adecuada de que un producto o servicio satisfará los requerimientos dados sobre calidad".

En Ingeniería de Software,

Según Roger S. Pressman: Consiste en la auditoria y las funciones de información de la gestión. El objetivo es proporcionar la gestión para informar de los datos necesarios sobre la calidad del producto, por lo que se va adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos.

4. Coste de Calidad

Son todos los costos acarreados en la búsqueda de la calidad o en las actividades relacionadas con la obtención de la calidad.

Se dividen en:

Costes de prevención: • Planificación de la calidad • Revisiones técnicas formales • Equipo de pruebas • Formación	**Costes de evaluación:** • Inspección en el proceso y entre procesos • Calibrado y mantenimiento del equipo • Pruebas
Costes de fallos internos • Retrabajo (revisión) • Reparación • Análisis de la modalidades de fallos	**Costes de fallos externos** • Resolución de quejas • Devolución y sustitución de productos • Soporte de línea de ayuda • Trabajo de garantía

1.2 La tendencia de la calidad

Se ha desarrollado una tendencia hacia la Gestión Total de Calidad (GTC), el cual comprende los siguientes cuatro pasos:

1

Kaizen

Se refiere a un sistema de mejora continua en el proceso. El objetivo es desarrollar un proceso que sea visible, repetible y mensurable.

El objetivo de la actitud Kaizen es el mejoramiento continuo en base a pequeños y constantes cambios, mediante la eliminación, reducción o cambio de las cosas, sistemas, medidas, etc.; que impiden un adecuado desempeño de las actividades.
En el marco empresarial, se traduce a que todos los miembros de una organización están comprometidos con la revisión constante de los procesos y la mejora permanente.

KAISEN está basado en las cinco "S"

SEIRE	Organización: Cada cosa en su lugar y un lugar para cada cosa.
SEITON	Reducir búsquedas: Facilitar el movimiento de las cosas, servicios y personas
SEISO	Limpieza: Cuando todo está limpio, todo está ordenado y se simplifican los procedimientos.
SEIKETSU	Estandarización y simplificación de procesos: Mantener el orden, organización y limpieza en el ambiente y las personas.
SHITSUKE	Disciplina y buenos hábitos de trabajo: Basados en el respeto a las reglas y a las personas (compañeros de trabajo y clientes)

2

Atarimae hinshitsu (Calidad Funcional)

Examina lo intangible que afecta al proceso y trabaja para optimizar su impacto en el proceso.

Calidad Funcional: es la calidad que ya se le asigna al producto o servicio mismo, si son autos... depende de la marca del auto para asignarle inconscientemente una calidad.

3

Kansei

Se centra en el usuario del producto. Examinando la forma en que el usuario aplica el producto, kansei conduce a la mejora en el producto mismo y potencialmente al proceso que lo creo.

Kansei es un término japonés donde la sílaba *kan* significa sensitividad y *sei* significa sensibilidad. Se usa para expresar la cualidad de un objeto de despertar placer en su uso. Así, hay objetos con mucho kansei y otros con absolutamente ninguno.
Cada vez más, se está expresando la necesidad de tener en cuenta los aspectos subjetivos (emoción, afecto, percepciones, sensaciones...) en la experiencia de uso, yendo más allá del puro diseño visual.

Las necesidades básicas que permitirán definir la estructura general del planteamiento Kansei son como sigue (Nagamachi, 1995):

- Obtener y cuantificar la respuesta del usuario en términos kansei (valoración psicosociológica).
- Identificar las características de diseño de un producto desde la percepción del usuario.
- Implementar la herramienta a partir de los datos anteriores.
- Ajustar el diseño del producto a los cambios sociales y a los que se producen en las preferencias de los usuarios con el paso del tiempo.

4

Miryoku Teki hinshitsu

Orientado a la gestión que busca la oportunidad en áreas relacionadas que se pueden identificar observando la utilización del producto en el mercado.

Calidad Emocional: es lo que te hace sentir el usar tal o cual producto o servicio.... no te sientes igual al conducir un Toyota que un mercedes Benz o que un auto volvo....

1.3 Garantía y aseguramiento de la calidad del software

La calidad del software se define como:

"La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario". (IEEE, Std. 610-1990).

"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente" (Pressman, 2002)

La garantía de calidad del software (**SQA**), comprende un conjunto de tareas y acciones sistemáticas y planificadas que permiten asegurar la calidad del software.

A este conjunto de tareas y acciones se le denomina **Actividades de SQA** y comprende:

Actividad	Características
Plan SQA para el proyecto	El plan debe involucrar: Evaluaciones, Auditorias, revisiones, estándares que se pueden aplicar al proyecto. Procedimientos para información, seguimiento de errores y realimentación. El grupo SQA debe además documentar la información necesaria.

Actividad	Características
Proceso de software del proyecto	Se determina el proceso y se realiza la revisión de la descripción del proceso para poder establecer los ajustes de acuerdo a las políticas de la organización.
Ajustes al proceso del software	El grupo SQA se encarga de revisar, documentar y verificar que se han hecho los ajustes al proceso.
Auditoria de los productos de software	El grupo SQA esta en constante revisión del proceso software e informa periódicamente los resultados al gestor del proyecto.
Documentación de productos software	Se debe documentar todas las desviaciones encontradas a nivel:

	• De procesos • De estándares y • Técnicos
Registro de ajustes a requisitos e informes	Se realiza seguimiento a los requisitos que no se ajustan y se elaboran los informes respectivos.

El grupo SQA, además de tener a cargo el plan SQA, también debe coordinar, control y gestionar los cambios, recopilar y analizar las métricas del software.

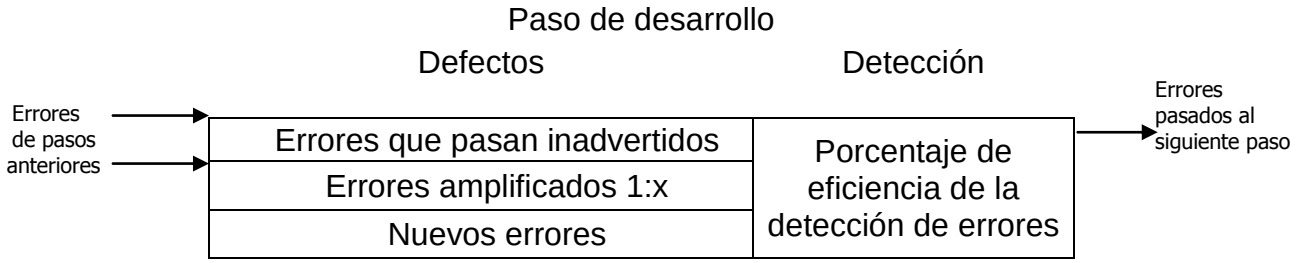
1.4 Revisiones del software

Las revisiones del software, son el conjunto de actividades que suceden como resultado del análisis, el diseño y la codificación y que sirven para depurar las actividades de ingeniería del software

Una revisión, tiene como objetivos:

- Señalar la necesidad de mejoras en el producto
- Continuar las partes de un producto en las que no es necesaria o no es deseable una mejora
- Conseguir un trabajo técnico de una calidad más uniforme, o al menos más predecible, que la que puede ser conseguida sin revisiones, con el fin de hacer más manejable el trabajo técnico.

Las revisiones de software se usan como modelo para la amplificación de defectos y para ilustrar la generación y detección de errores durante los pasos de diseño preliminar, diseño detallado y codificación del proceso de ingeniería del software.



En cada paso del proceso de desarrollo de software, se presentan errores que pasan inadvertidos y que producen un mayor número de errores si las revisiones no lo detectan.
Los errores amplificados corresponden, a aquellos errores que pasan inadvertidos desde pasos anteriores. De igual forma se representa el porcentaje de eficiencia de la detección de errores.

Revisiones técnicas formales

Una revisión técnica formal (RTF) es un medio efectivo para mejorar la calidad del software.

Los objetivos de la RTF son:

- Descubrir errores en la función, la lógica o la implementación de cualquier representación del software
- Verificar que el software bajo revisión alcanza sus requisitos
- Garantizar que el software ha sido representado de acuerdo con ciertos estándares predefinidos
- Conseguir un software desarrollado de forma uniforme
- Hacer que los proyectos sean manejables

La RTF incluye:

- Recorridos
- Inspecciones
- Revisiones cíclicas
- Evaluaciones técnicas del software

Cada RTF debe estar debidamente planificada, controlada y atendida por el grupo encargado de cada RTF.

Cada reunión debe tener las siguientes características:

- Deben convocarse para la revisión entre tres y cinco personas
- Se debe preparar por adelantado,
- La duración de la reunión de revisión, debe ser menor de dos horas

¿Quienes participan en la reunión?

- El jefe de revisión: quien lidera la reunión
- Los revisores: uno de los cuales se encarga de registrar todos los sucesos de la reunión.
- El productor

Registro e informe de la revisión

Como se menciono anteriormente, uno de los revisores es el encargado de registrar todos los acontecimientos y conclusiones que van surgiendo durante la RTF.

Al final de la reunión, hace un resumen de las conclusiones y genera una lista de sucesos de revisión. Además, prepara un informe sumario de la revisión técnica formal que responda a las siguientes preguntas:

¿Que fue revisado**?**
¿Quién lo revisó**?**
¿Qué se descubrió y cuáles son las conclusiones**?**

La lista de sucesos de revisión que se genera permite:

- Identificar áreas problemáticas dentro del producto
- Servir como lista de comprobación para hacer las correcciones.

Además se adjunta, la lista de conclusiones al informe sumario.

Directrices para la revisión

1. Revisar el producto, no al productor	Se deben señalar los errores de forma constructiva y no dificultar el proceso de revisión. Es importante mantener el control de la reunión y descartar situaciones que se escapen de control.
2. Fijar una agenda y mantenerla	Se debe tener un plan de trabajo antes de la reunión. Se debe seguir el orden del plan para que la reunión tenga éxito y cumplir con los tiempos asignados al

	plan.
3. Limitar el debate y las impugnaciones	No se debe perder tiempo debatiendo situaciones que no presenten unanimidad, es importante registrar el hecho y dedicar otros tiempos para su debate.
4. Enunciar áreas problemas pero no intentar resolver los problemas que se pongan de manifiesto.	La resolución de problemas se debe programar para otros espacios después de la reunión de revisión.
5. Tomar notas escritas	Es buena idea utilizar diferentes herramientas para la toma de notas, por ejemplo, pizarras, tableros, computador, para que se pueda hacer seguimiento a la asignación de prioridades.
6. Limitar el número de participantes e insistir en la preparación anticipada	Se debe limitar el número de revisores, los cuales deben estar preparados para cada reunión y participar activamente en el proceso de revisión.
7. Desarrollar una lista de comprobación para cada producto que haya de ser revisado.	Se deben desarrollar listas de comprobación para los documentos de análisis, diseño, codificación y pruebas.
8. Disponer recursos y una agenda para las RTF	Cada RTF debe estar planificada e involucrar modificaciones.
9. Capacitación y entrenamiento de los revisores	Todas las personas que participen en el proceso de revisión deben recibir un entrenamiento que se debe basar en: • El proceso • Psicología humana
10. Revisar las revisiones anteriores	Son beneficiosas porque permiten descubrir problemas del proceso de revisión.

1.5 Garantía de calidad estadística

La garantía de calidad estadística, permite establecer la calidad más cuantitativamente.

Para el software, comprende los siguientes pasos:

1. Agrupar y clasificar la información sobre los defectos del software.
2. Encontrar la causa de cada defecto
3. Mediante el principio de Pareto (el 80 por 100 de los defectos se pueden encontrar en el 20 por 100 de todas las posibles causas), se aísla el 20 por 100 (los «pocos vitales»).

   El nombre de Pareto fue dado por el Dr. Joseph Juran en honor del economista italiano Vilfredo Pareto (1848-1923) quien realizo un estudio sobre la distribución de la riqueza, en el cual descubrió que la minoría de la población poseía la mayor parte de la riqueza y la mayoría de la población poseía la menor parte de la riqueza. Con esto estableció la llamada "Ley de Pareto". También se conoce como la regla 80/20.
   Según este concepto, si se tiene un problema con muchas causas, podemos decir que el 20% de las causas resuelven el 80% del problema y el 80% de las causas solo resuelven el 20% del problema. Por lo tanto, el Análisis de Pareto es una técnica que separa los "pocos vitales" de los "muchos triviales".
   Para ampliar y profundizar en esta ley, puede consultar: http://www.gestiopolis.com/recursos/documentos/fulldocs/eco/diagramapareto.htm

4. Una vez que se han identificado los defectos vitales, se actúa para corregir los problemas que han producido los defectos.

El siguiente es un ejemplo de aplicación:

Los siguientes son los defectos mas frecuentes que se han encontrado en procesos de desarrollo de software:

- Especificación incompleta o errónea
- Mala interpretación de la comunicación del cliente
- Desviación deliberada de la especificación
- Incumplimiento de los estándares de programación
- Error en la representación de los datos
- Interfaz de módulo inconsistente
- Error en la lógica de diseño
- Prueba incompleta o errónea
- Documentación imprecisa o incompleta
- Error en la traducción del diseño al lenguaje de programación
- Interfaz hombre-maquina ambigua o inconsistente
- Varios

Posteriormente, un inspector revisa y registra los defectos de acuerdo con dichos tipos. Después de inspeccionar 942 errores, se obtuvo una tabla como la siguiente:

Error	Frecuencia (No.)
Especificación incompleta o errónea	205
Mala interpretación de la comunicación del cliente	156
Desviación deliberada de la especificación	48
Incumplimiento de los estándares de programación	25
Error en la representación de los datos	130
Interfaz de módulo inconsistente	58
Error en la lógica de diseño	45
Prueba incompleta o errónea	95
Documentación imprecisa o incompleta	36
Error en la traducción del diseño al lenguaje de programación	60
Interfaz hombre-maquina ambigua o inconsistente	28
Varios	56
Total	942

El siguiente paso es utilizar la frecuencia porcentual es decir, el porcentaje de errores en cada tipo de defecto:

Error	Frecuencia (No.)	Frec. %
Especificación incompleta o errónea	205	22%
Mala interpretación de la comunicación del cliente	156	17%
Desviación deliberada de la especificación	48	5%
Incumplimiento de los estándares de programación	25	3%
Error en la representación de los datos	130	14%
Interfaz de módulo inconsistente	58	6%
Error en la lógica de diseño	45	5%
Prueba incompleta o errónea	95	10%
Documentación imprecisa o incompleta	36	4%
Error en la traducción del diseño al lenguaje de programación	60	6%
Interfaz hombre-maquina ambigua o	28	3%

inconsistente		
Varios	56	6%
Total	942	100%

El objetivo es determinar cuáles son los defectos que aparecen con mayor frecuencia. Para esto se ordenan los datos de la tabla en orden decreciente de frecuencia:

Error	Frecuencia (No.)	Frec. %
Especificación incompleta o errónea	205	22%
Mala interpretación de la comunicación del cliente	156	17%
Error en la representación de los datos	130	14%
Prueba incompleta o errónea	95	10%
Interfaz de módulo inconsistente	58	6%
Error en la traducción del diseño al lenguaje de programación	60	6%
Desviación deliberada de la especificación	48	5%
Error en la lógica de diseño	45	5%
Documentación imprecisa o incompleta	36	4%
Incumplimiento de los estándares de programación	25	3%
Interfaz hombre-maquina ambigua o inconsistente	28	3%
Varios	56	6%
Total	942	100%

De esta forma, resulta evidente cuales son los tipos de defectos más frecuentes. Podemos observar que los 4 primeros tipos de defectos representan más del 60%. Por el Principio de Pareto, se puede concluir que: La mayor parte de los defectos encontrados pertenece sólo a 4 tipos de defectos, de manera que si se eliminan las causas que los provocan desaparecería la mayor parte de los defectos.

Fiabilidad del software

La fiabilidad del software es la probabilidad de que en un tiempo y entorno determinado el software en operación este libre de fallos.

Los fallos del software, se producen por problemas de diseño o de implementación.
La medida de fiabilidad del software está dada por:

- El tiempo medio entre fallos (TMEF),

$$TMEF = TMDF + TMDR$$

Donde:

TMDF	Tiempo medio de fallo
TMDR	Tiempo medio de reparación

- La disponibilidad del software

Es la probabilidad de que un programa funcione de acuerdo con los requisitos en un momento dado.

Se define como:

$$\text{Disponibilidad} = \frac{TMDF}{TMDF+TMDR}(100\%)$$

Estos datos pueden ser medidos o estimados mediante datos históricos o de desarrollo.

El estándar de calidad ISO 9001

La Organización Internacional para la Estandarización (ISO) es una federación mundial de cuerpos de normas nacionales de aproximadamente 140 países. ISO es una organización establecida en 1947, cuya misión es promover el desarrollo de la estandarización y de las actividades relacionadas en el mundo, con la idea de que

facilita el cambio internacional de bienes y servicios, y la cooperación que se desarrolla en las esferas de la actividad intelectual, la actividad científica, tecnológica y económica.

ISO 9001. Quality Systems- Model for Quality Assurance in desing, development, production, installation and servicing. Este es un estándar que describe el sistema de calidad utilizado para mantener el desarrollo de un producto que implique diseño.

La norma ISO 9001, son un conjunto de reglas de carácter social y organizativo para mejorar y potenciar las relaciones entre los miembros de una organización. Cuyo último resultado, es mejorar las capacidades y rendimiento de la organización, y conseguir un aumento por este procedimiento de la calidad final del producto.

Los 8 Principios básicos de la gestión de la calidad o excelencia son:

1. Organización enfocada a los clientes
   Las organizaciones dependen de sus clientes y por lo tanto comprender sus necesidades presentes y futuras, cumplir con sus requisitos y esforzarse en exceder sus expectativas.
2. Liderazgo
   Los lideres establecen la unidad de propósito y dirección de la organización. Ellos deben crear y mantener un ambiente interno, en el cual el personal pueda llegar a involucrarse totalmente para lograr los objetivos de la organización.
3. Compromiso de todo el personal
   El personal, con independencia del nivel de al organización en el que se encuentre, es la esencia de la organización y su total implicación posibilita que sus capacidades sean usadas para el beneficio de la organización.
4. Enfoque a procesos
   Los resultados deseados se alcanzan más eficientemente cuando los recursos y las actividades relacionadas se gestionan como un proceso.
5. Enfoque del sistema hacia la gestión
   Identificar, entender y gestionar un sistema de procesos interrelacionados para un objeto dado, mejora la eficiencia y la eficiencia de una organización.
6. La mejora continua
   La mejora continua debería ser el objetivo permanente de la organización.
7. Enfoque objetivo hacia la toma de decisiones
   Las decisiones efectivas se basan en el análisis de datos y en la información.
8. Relaciones mutuamente beneficiosas con los proveedores
   Una organización y sus proveedores son independientes y una relación mutuamente benéfica

ISO 9004-2. Quality Management and Quality System Elements —Part 2—. Este documento proporciona las directrices para el servicio de facilidades del software como soporte de usuarios.

La norma para "Gestión de calidad y elementos del sistema de calidad. Parte 2: Directrices para servicios" constituye una de las normas ISO de la serie 9000 relacionadas con la gestión y el aseguramiento de la calidad en diferentes tipos de organizaciones.

La satisfacción de las expectativas de los clientes, así como el logro y el mantenimiento de la calidad deseada por los mismos, son metas deseables para cualquier organización. Sin embargo, estos aspectos tienen un interés social más claro cuando la organización se dedica a la prestación de servicios.

Estructura

Características de los servicios

- Características del servicio y de la prestación del servicio
- Control de las características del servicio y de la prestación del servicio

Principios del sistema de calidad

- Aspectos clave de un sistema de calidad
- Responsabilidad gerencial
- Personal y recursos materiales
- Estructura del sistema de calidad
- Interfase con los clientes

Elementos operaciones del sistema de calidad

- Proceso de comercialización
- Proceso de diseño
- Proceso de prestación del servicio
- Análisis y mejoramiento del comportamiento del servicio

- General.
- Análisis del contrato
- Especificación de los requisitos del comprador
- Planificación del desarrollo
- Planificación de la calidad
- Proyecto e implementación
- Pruebas y validaciones
- Aceptación
- Reproducción, entrega e instalación
- Mantenimiento
- Sistema de la calidad - actividades de apoyo (independientes de cualquier fase)
- Gestión de la configuración
- Control de documentos
- Registros de la calidad
- Medición
- Reglas, prácticas y convenciones
- Herramientas y técnicas
- Aprovisionamiento
- Productos de software incluidos

ACTIVIDADES COMPLEMENTARIAS

1. Investigue y elabore un ensayo argumentativo sobre la evolución histórica de la Calidad.

2. ¿Es posible evaluar la calidad del software si el cliente no se pone de acuerdo sobre lo que se supone que ha de hacer? Justifique su respuesta y argumente.

3. La calidad y la fiabilidad son conceptos relacionados pero son fundamentalmente diferentes en varias formas. Elabore un cuadro de diferencias.

4. Si se le da la responsabilidad de mejorar la calidad del software en su organización. ¿Qué es lo primero que haría? ¿Qué sería lo siguiente?

5. Una reunión de revisión técnica formal sólo es efectiva si todo el mundo se prepara por adelantado. ¿Cómo descubriría que uno de los participantes no se preparado? ¿Qué haría si fuera el jefe de división?

6. Investigue y amplíe la información de ISO 9000-3. Elabore un ensayo.

2. TÉCNICAS DE PRUEBA DEL SOFTWARE

Las pruebas del software comprenden el examen o exploración final de las especificaciones del diseño y de la codificación.

Las pruebas del software son un conjunto de evaluaciones cuyo fin es identificar y descubrir un error.

Las técnicas de pruebas del software permiten diseñar pruebas que:

1. Comprueben la lógica interna de los componentes software
2. Verifiquen los dominios de entrada y salida del programa para descubrir errores en la funcionalidad, el comportamiento y rendimiento.

El encargado de realizar las pruebas es el Ingeniero de software con ayuda de especialistas en pruebas.

El software debe aprobarse desde dos perspectivas:

1. La lógica interna del programa. Utilizando pruebas de "caja blanca"
2. Los requisitos del software. Utilizando pruebas de caja negra

2.1 Fundamentos de la prueba del software

Las pruebas de software tienen los siguientes objetivos:

- Descubrir un error
- Mostrar un error no descubierto hasta ese momento
- Descubrir un error no detectado hasta ese momento

Las pruebas tienen los siguientes principios:

- Las pruebas deben tener un seguimiento hasta los requisitos del cliente.
- Las pruebas deben planificarse antes de que empiecen.
- Es aplicable el principio de Pareto a la prueba del software.
- No es posible las pruebas exhaustivas
- Las pruebas deben ser realizadas por un equipo independiente

Un software debe ser fácil de probar, para lo cual se puede tener en cuenta las siguientes características que propone Pressman:

Característica	Observación
Operatividad	Cuánto mejor funcione, más eficientemente se puede probar • El sistema tiene pocos errores • Ningún error bloquea la ejecución de las pruebas • El producto evoluciona en fases funcionales
Observabilidad	Lo que se ve es lo que se prueba • Se genera una salida distinta para cada entrada • Todos los factores que afectan a los resultados están visibles • Un resultado incorrecto se identifica fácilmente • Los errores internos se detectan automáticamente • Se informa automáticamente de los errores internos • El código fuente es accesible.
Controlabilidad	Cuánto mejor se pueda controlar el software, más se puede automatizar y optimizar • Todo el código es ejecutable a través de alguna combinación de entrada • El ingeniero de pruebas puede controlar los estados y las variables del hardware y software • Los formatos de las entradas y los resultados son consistentes y estructurados
Capacidad de descomposición	Controlando el ámbito de las pruebas, podemos aislar más rápidamente los problemas y llevar a cabo pruebas de regresión • El software esta construido con módulos independientes • Los módulos de software se pueden probar independientemente.
Simplicidad	Cuánto menos haya que probar, más rápidamente se puede probar • Simplicidad funcional • Simplicidad estructural • Simplicidad del código

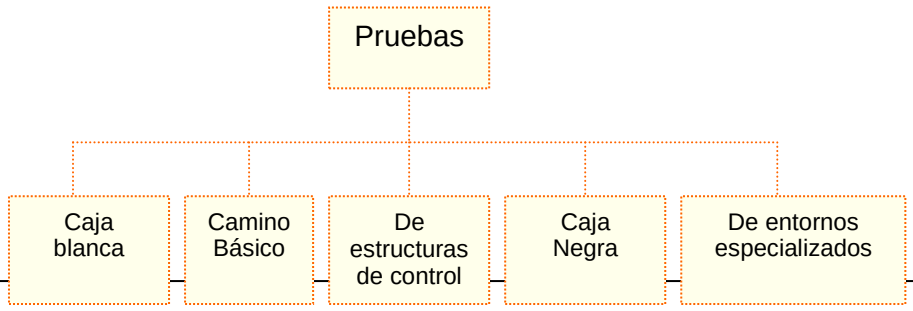
Característica	Observación
Estabilidad	Cuanto menos cambios, menos interrupciones a las pruebas • Los cambios del software son infrecuentes • Los cambios del software están controlados • Los cambios del software no invalidan las pruebas existentes • El software se recupera bien de los fallos
Facilidad de comprensión	Cuanta más información se tenga, más inteligentes eran las pruebas • El diseño se ha entendido perfectamente • Las dependencias entre los componentes internos, externos y compartidos se han entendido perfectamente • Se han comunicado los cambios del diseño • La documentación técnica es accesible

2.2 Diseño de casos de prueba

El diseño de casos de prueba, tiene un único objetivo: tener la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo posible.

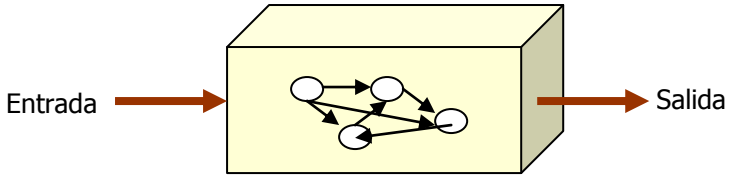
Cualquier producto software puede aprobarse de una las siguientes formas:

1. Conociendo la función para la que fue diseñado el producto.
   - Se pueden utilizar pruebas para: comprobar su función operativa y buscar errores de cada función.
2. Conociendo el funcionamiento del producto.
   - Se pueden utilizar pruebas para: comprobar que las operaciones esta de acuerdo con las especificaciones y para comprobar que los componentes internos funcionan de forma adecuada.



Prueba de caja blanca

Esta prueba se centra en la estructura interna del programa. En este caso la prueba consiste en probar todos los posibles caminos de ejecución a través de las instrucciones del código, que puedan trazarse.



Mediante está prueba, el ingeniero del software puede:

1. Garantizar que se recorre por lo menos una vez todos los caminos independientes de cada módulo.
2. Recorrer todas las decisiones lógicas en sus condiciones verdadera y falsa.
3. Recorrer todos los bucles en sus límites y con sus límites operacionales.
4. Recorrer las estructuras internas de datos para asegurar su validez

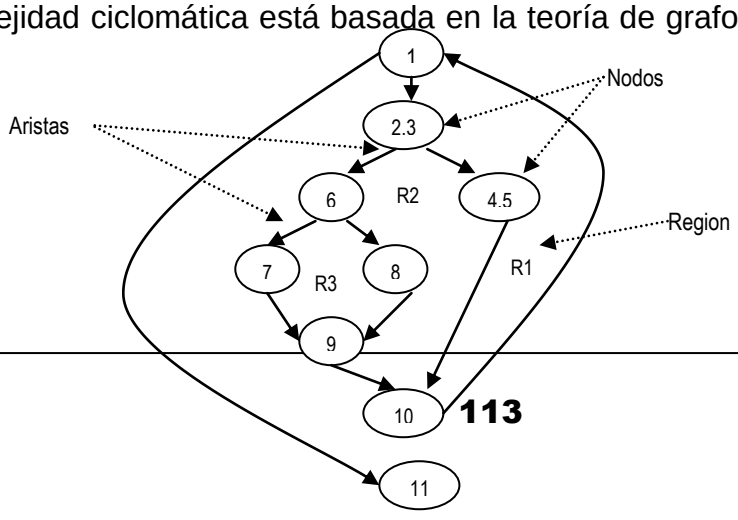
Prueba del camino básico

Esta prueba permite obtener una medida de la complejidad de la lógica de un diseño procedimental y usar ésa medida como guía para la definición de un conjunto básico de camino de ejecución. Esta prueba permite que se ejecute por lo menos una vez cada sentencia del programa.

Complejidad ciclomática

Es una métrica que proporciona una medición cuantitativa de la complejidad lógica de un programa.

La complejidad ciclomática está basada en la teoría de grafos por lo que es importante recordar:



La complejidad ciclomática se calcula de las siguientes formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática
2. La complejidad ciclomática, V(G) de un grafo de flujo G se define como:

Donde:

A	Es el número de aristas del grafo de flujo
N	Es el número de nodos del grafo de flujo

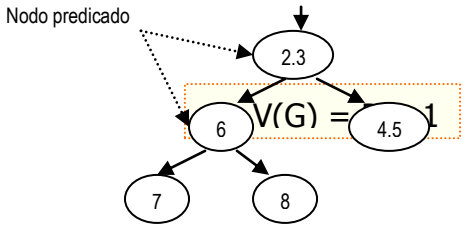
3. La complejidad ciclomática, V(G), de un grafo de flujo G también se define:

Donde:

$$V(G) = A - N + 2$$

P	Es el número de nodos predicado contenidos en el grafo de flujo G

Un nodo predicado: es cada nodo que contiene una condición.



Entonces,

V(G) = 11 aristas - - 9 nodos + 2 = 4
o,
V(G)= 3 nodos predicado + 1 = 4

2.3 Prueba de la estructura de control

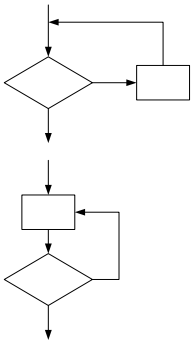
Comprende las siguientes pruebas:

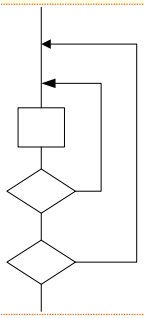
1. Prueba de condición
   Se centra en la prueba de cada una de las condiciones del programa y tiene como propósito detectar los errores en las condiciones de un programa y los errores del programa.

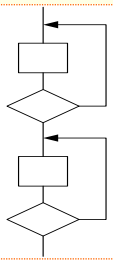
2. Prueba del flujo de datos

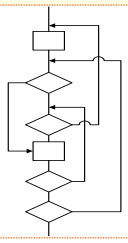
Se centra en la selección de caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
Esta prueba es útil para seleccionar caminos de prueba de un programa que contenga sentencias if o bucles anidados.

3. Prueba de bucles
   Se centra en la validez de las construcciones de bucles. Se definen los siguientes tipos de bucles:

Bucles Simples	
	Se debe aplicar: • Pasar por alto totalmente el bucle • Pasar una sola vez por el bucle • Pasar dos veces por el bucle • Hacer m pasos por el bucle con m < n • Hacer n-1, n y n+1 pasos por el bucle Donde n, es el número máximo de pasos permitidos por el bucle.

Bucles Anidados	
	Se debe: • Comenzar por el bucle más interior • Llevar a cabo las pruebas de bucles simples para el bucle más interior • Progresar hacia fuera, llevando a cabo pruebas para el siguiente bucle • Continuar hasta cuando se prueben todos los bucles.

Bucles Concatenados	
	Se pueden probar con el método para buques simples, siempre y cuando los bucles sean independientes. Cuando los bucles no son independientes se utiliza el enfoque para bucles anidados.

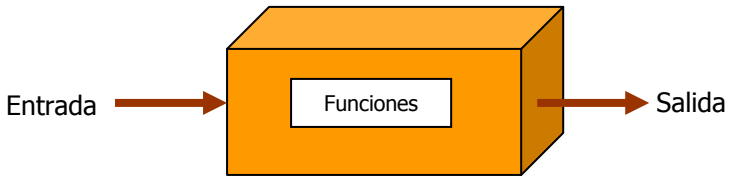
Bucles no estructurados	
	Estos bucles se deben rediseñar.

2.4 Prueba de caja negra

Consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos. Aquí, la prueba ideal del software consiste en probar todas las posibles entradas y salidas del programa.

La prueba de caja negra, también encuentra errores de:

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación



2.5 Prueba de entornos especializados, arquitecturas y aplicaciones

Prueba de interfaces gráficas de usuario

Se utilizan listas de chequeo:

- Para ventanas:
  - ¿Se abren las ventanas mediante órdenes basadas en el teclado o en un menú?
  - ¿Se puede ajustar el tamaño, mover y desplegar la ventana?
  - ¿Se regenera adecuadamente cuando se escribe y se vuelve a abrir?

Para menús emergentes y operaciones con el ratón:

- ¿Se muestra la barra de menú apropiada en el contexto apropiado?
- ¿Es correcto el tipo, tamaño y formato del texto?
- ¿Si el ratón tiene varios botones, están apropiadamente reconocidos en el contexto?

Entrada de datos:

- ¿Se repiten y son introducidos adecuadamente los datos alfanuméricos?
- ¿Funcionan adecuadamente los modos gráficos de entrada de datos? (p.e. barra deslizante)
- ¿Se reconocen adecuadamente los datos no válidos?
- ¿Son inteligibles los mensajes de entrada de datos?

Prueba de arquitectura cliente / servidor

Debido a la complejidad del sistema, serán necesarias varias fases:

- Pruebas de funcionalidad de la aplicación. Se puede llevar a cabo sobre máquinas de desarrollo y estaciones de trabajo de forma paralela
- Pruebas de carga del servidor
- Pruebas de integridad de datos: Son especialmente importantes en el caso de bases de datos distribuidas
- Pruebas transaccionales
- Pruebas de red

Prueba de la documentación y facilidades de ayudas

Se puede dar en dos sentidos:

- Revisión e inspección: examina la documentación para comprobar la claridad de la misma.
- Prueba en vivo: se utiliza la documentación junto al uso del software.

Prueba de sistemas de tiempo real

Se puede aplicar los siguientes pasos:

- Prueba de tareas: Se aplican pruebas de caja blanca y caja negra a cada tarea. Pretende descubrir errores en la lógica y en el funcionamiento.
- Prueba de comportamiento: Se simula el comportamiento del sistema en tiempo real y se examina el comportamiento como consecuencia de sucesos externos.
- Prueba intertareas: Se prueban las tareas asíncronas que se comunican con otras, para determinar si se producen errores de sincronismo entre las tareas.
- Prueba del sistema: Se realizan pruebas completas al sistema para descubrir errores en la interfaz software/hardware.

ACTIVIDADES COMPLEMENTARIAS

1. Investigue y amplíe la información relacionada con:

    - Prueba de interfaces gráficas de usuario
    - Prueba de arquitectura cliente / servidor
    - Prueba de la documentación y facilidades de ayudas
    - Prueba de sistemas de tiempo real

2. Investigue y amplíe la información relacionada con:

    - Prueba de condición
    - Prueba del flujo de datos
    - Prueba de bucles

3. [3]ESTRATEGIAS DE PRUEBA DEL SOFTWARE

La estrategia proporciona la descripción de los pasos que hay que llevar a cabo como parte de la prueba, cuándo se deben planificar y realizar esos pasos, y cuánto esfuerzo, tiempo y recursos se van a requerir.

Una estrategia de prueba contiene:

- Planificación de la prueba
- Diseño de casos de prueba
- Ejecución de las pruebas
- Agrupación y evaluación de datos

3.1 Enfoque estratégico para las pruebas del software

Según Roger S. Pressman, las pruebas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón, se define una plantilla para las pruebas del software.

Una estrategia de prueba del software tiene las siguientes características generales:

- Las pruebas comienzan a nivel de módulo (en los sistemas orientados a objetos, las pruebas empiezan a nivel de clase o de objeto) y trabajan "hacia fuera", hacia la integración de todo el sistema.
- Según el momento, son apropiadas diferentes técnicas de prueba
- La prueba la lleva a cabo el responsable del desarrollo del software y (para grandes proyectos) un grupo independiente de pruebas.
- La prueba y la depuración son actividades diferentes, pero la depuración se debe incluir en cualquier estrategia de prueba

Una estrategia de prueba de software proporciona una guía al profesional y proporciona un conjunto de hitos para el jefe de proyecto.

[3] Ingeniería del software. Un enfoque práctico. Roger S. Pressman. 2002

3.1.1 Verificación y validación

Verificación
Es el conjunto de actividades que aseguran que el software implementa correctamente una función específica. ¿Estamos construyendo el producto correctamente?

Validación
Es el conjunto de actividades diferentes que aseguran que el software construido se ajusta a los requisitos del cliente. ¿Estamos construyendo el producto correcto?

3.1.2 Organización para las pruebas del software

Toda prueba de software debe tener la coordinación de actividades:

- El responsable del desarrollo del software es el responsable de probar las unidades del programa y a veces se encarga también de la prueba de integración.

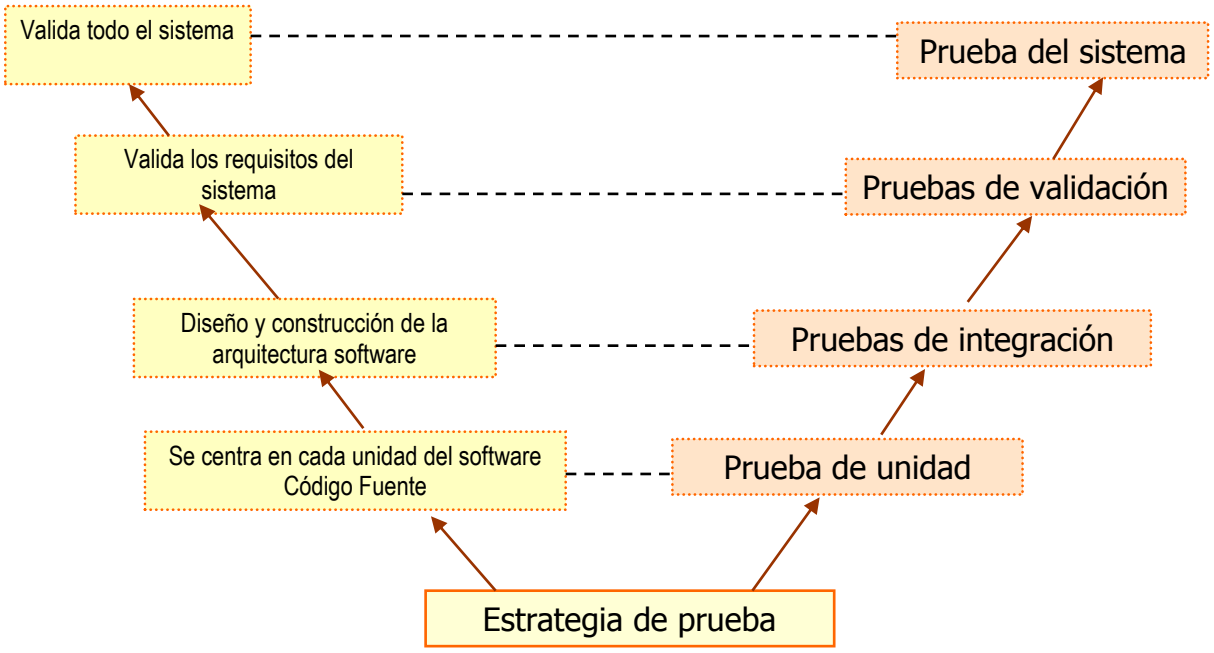
- Cuando se tiene una arquitectura completa de software, los encargados de la prueba es un *Grupo Independiente de Prueba* (GIP), permitiendo que se tenga independencia.

Grupo Independiente de prueba

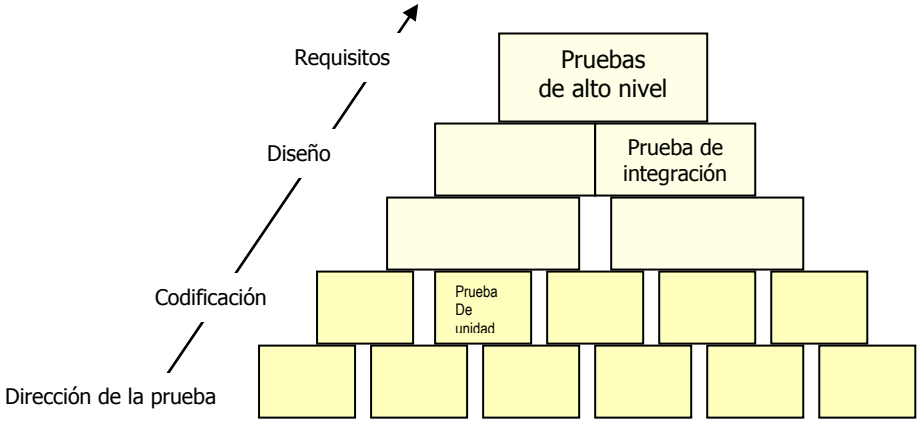
Este grupo tiene como objetivo identificar y encontrar errores. Este grupo trabaja conjuntamente con el responsable del desarrollo de software para asegurar que se realizan pruebas exhaustivas. Mientras se realiza la prueba, el desarrollador debe estar disponible para corregir los errores que se van descubriendo.

3.1.3 Estrategia de prueba del software

Los niveles de la estrategia para la prueba del software se pueden ver en el siguiente grafico:



Si se considera el proceso desde el punto de vista procedimental, la prueba, en el contexto de la ingeniería del software, realmente es una serie de cuatro pasos que se llevan a cabo secuencialmente.



Etapas en la prueba del software (Preesman, 2005)

1. La prueba se centra en cada módulo individualmente, asegurando que funcionan adecuadamente como una unidad. La prueba de unidad hace uso de las técnicas de prueba de caja blanca, ejercitando caminos específicos de la estructura de control del módulo para asegurar un alcance completo y una detección máxima de errores.

2. Se ensamblan o integran los módulos para formar el paquete de software completo. La prueba de integración se dirige a todos los aspectos asociados con el doble problema de verificación y de construcción del programa. Durante la integración, las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra, aunque se pueden llevar a cabo algunas pruebas de caja blanca con el fin de asegurar que se cubren los principales caminos de control.

3. Después de que el software se ha integrado (construido), se dirigen un conjunto de pruebas de alto nivel. Se deben comprobar los criterios de validación. La prueba de validación proporciona una seguridad final de que el software satisface todos los requisitos funcionales, de comportamiento y de rendimiento. Durante la validación se usan exclusivamente técnicas de prueba de caja negra.

4. La prueba del sistema verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total.

Si se desea implementar una estrategia de software con éxito se debe plantea que se deben abordar los siguientes si se desea implementar con éxito una estrategia de prueba del software se debe tener presente:

Especificar los requisitos del producto de manera cuantificable mucho antes de que comiencen las pruebas

También se debe evaluar: portabilidad, facilidad de mantenimiento y facilidad de uso

Establecer los objetivos de la prueba de manera explícita

Se debe establecer:

- Objetivos específicos de la prueba
- Cobertura de la prueba
- Tiempo medio de fallo
- El coste para encontrar y arreglar errores
- Densidad de fallos remanente o frecuencia de ocurrencia
- Horas de trabajo por prueba

Comprender qué usuarios van a manejar el software y desarrollar un perfil para cada categoría de usuario

Se debe:

- Describir el escenario de interacción para cada clase de usuario

Desarrollar un plan de prueba que haga hincapié en la "prueba de ciclo rápido"

El equipo de ingeniería de software, debe aprender a probar en ciclos rápidos y que se pueda probar sobre el terreno.

Construir un software "robusto" diseñado para probarse a sí mismo.

El software debe ser capaz de diagnosticar ciertas clases de errores. Además, el diseño debe incluir pruebas automatizadas y pruebas de regresión.

Usar revisiones técnicas formales efectivas como filtro antes de la prueba

Las revisiones técnicas formales ayudan a reducir el esfuerzo de prueba necesaria para la producción del software.

Llevar a cabo revisiones técnicas formales para evaluar la estrategia de prueba y los propios casos de prueba.

Permiten descubrir inconsistencias, omisiones y errores claros en el enfoque de la prueba.

Desarrollar un enfoque de mejora continua al proceso de prueba. Debería medirse la estrategia de prueba.

Permite usar un enfoque estadístico de control del proceso para la prueba del software.

3.2 Prueba de unidad

El proceso de verificación, se centra en la menor unidad del diseño del software: el módulo.

Está orientada a caja blanca y este paso se puede llevar a cabo en paralelo para múltiples módulos. Las pruebas que se dan como parte de la prueba de unidad son:

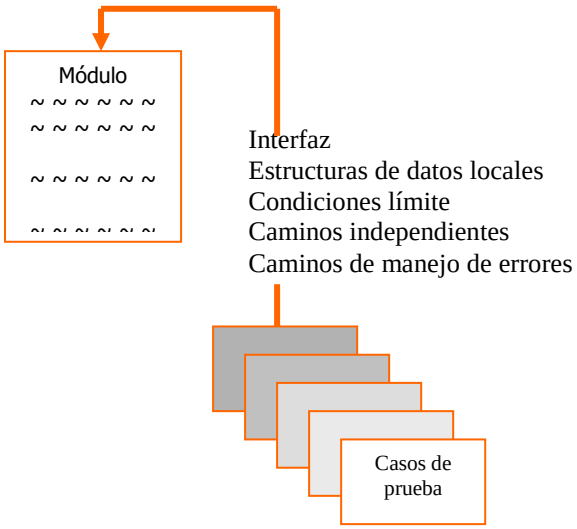


Figura. Prueba de Unidad. (Fuente: Roger Pressman, 2002)

Prueba de interfaz del módulo	Asegura que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada.
Prueba de estructuras de datos locales	Asegura que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo.
Prueba de condiciones de límite	Asegurar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento.
Prueba de caminos independientes	Se recorren los caminos independientes de la estructura de control con el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez.
Prueba de camino de manejo de errores.	Se prueban todos los caminos.

A continuación, se ilustra el entorno para la prueba de unidad.

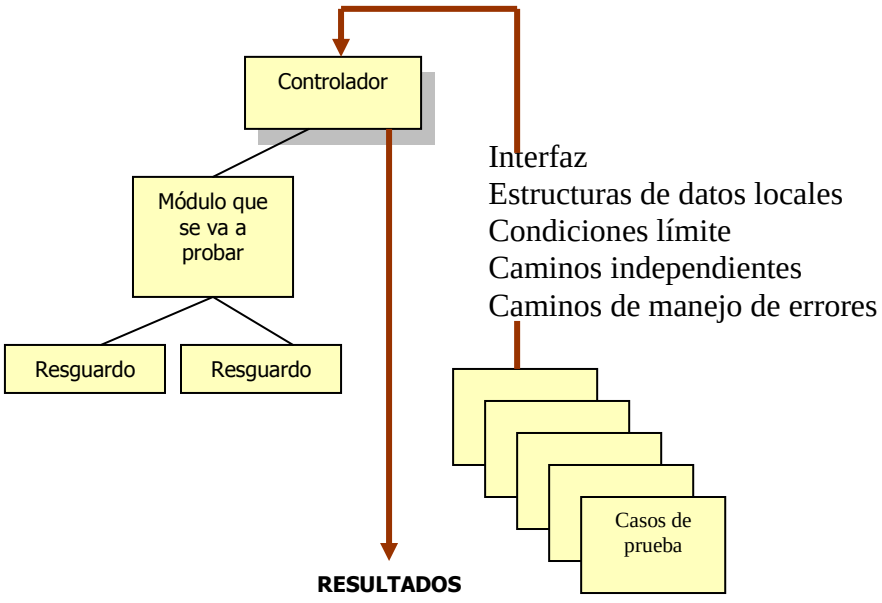


Figura. Entorno para la prueba de unidad

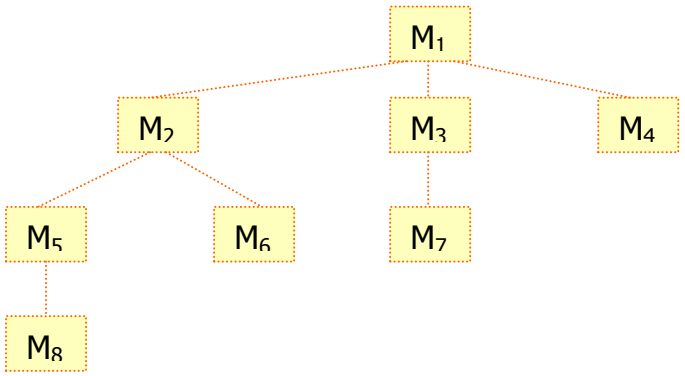
Para cada módulo que se va a probar, se crea un controlador (un programa principal) que permite aceptar los datos del caso de prueba, los pasa al módulo y luego imprime los resultados.

3.3 Prueba de integración del sistema

La prueba de integración es una técnica para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción.

3.3.1 Integración descendente

Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando por el módulo de control principal (programa principal). Los módulos subordinados al módulo de control principal se van incorporando en la estructura, bien de forma primero en profundidad, o bien de forma primero en anchura.



Integración descendente

- Integración primero en profundidad: integra todos los módulos de un camino de control principal de la estructura.
  Por ejemplo, si se elige el camino de la izquierda se integrarán primero los módulos M1, M2 y M5. A continuación, se integrará M8 o M6. A continuación se construyen los caminos de control central y derecho.

- Integración primero en anchura: incorpora todos los módulos directamente subordinados a cada nivel, moviéndose por la estructura de forma horizontal.
  Por ejemplo: Los primeros módulos que se integran son M2, M3 y M4. A continuación, sigue el siguiente nivel de control, M5, M6, M7 y por último M8.

El proceso de integración se realiza en cinco pasos:

1. Se usa el módulo de control principal como controlador de la prueba, disponiendo de resguardos para todo los módulos directamente subordinados al módulo de control principal.
2. Dependiendo del enfoque de integración elegido se van sustituyendo uno a uno los resguardos subordinados por los módulos reales.

3. Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.
4. Tras terminar cada conjunto de prueba, se reemplaza otro resguardo con el módulo real.
5. Se hace la prueba de regresión para asegurarse de que no se han introducido errores nuevos.

El proceso continúa desde el paso dos hasta que se haya construido la estructura del programa entero.

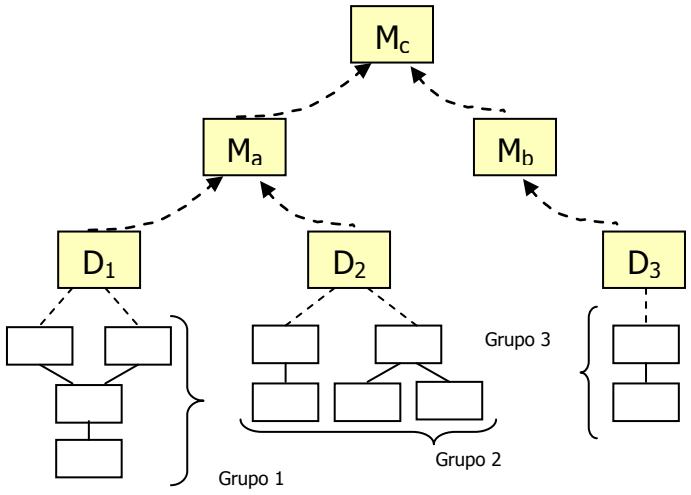
3.3.2 Integración ascendente

Empieza la construcción y la prueba con los niveles más bajos de la estructura del programa. Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardos.

Se puede implementar una estrategia de integración ascendente mediante los siguientes pasos:

1. Se combina los módulos de bajo nivel en grupos que realicen una subfunción específica del software.
2. se describe un controlador (un programa de control de la prueba) para coordinar la entrada y la salida de los casos de prueba.
3. Se prueba el grupo.
4. Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

La integración sigue el esquema de la siguiente figura:



Integración ascendente

Se combinan los módulos para formar los grupos 1,2 y 3. Cada uno de los grupos se somete a prueba mediante un controlador (mostrado como un bloque punteado). Los módulos de los grupos 1 y 2 son subordinados M_a. Los controladores D_1 y D_2 se eliminan y los grupos interaccionan directamente con M_a. De forma similar, se elimina el controlador D_3 del grupo 3 antes de la integración con el módulo M_b. Tanto M_a como M_b se integraran finalmente con el módulo M_c y así sucesivamente.

3.3.3 Prueba de regresión

La prueba de regresión consiste en volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios no han propagado efectos colaterales no deseados. La prueba de regresión es la actividad que ayuda a asegurar que los cambios no introducen un comportamiento no deseado o errores adicionales.

El conjunto de pruebas de regresión contiene tres clases diferentes de casos de prueba:

- Una muestra representativa de pruebas que ejercite todas las funciones del software;
- Pruebas adicionales que se centran en las funciones del software que se van a ver probablemente afectadas por el cambio;
- Pruebas que se centran en los componentes del software que han cambiado.

3.3.4 Prueba de humo

Esta prueba es utilizada cuando se ha desarrollado un producto software "empaquetado". Es diseñado como un mecanismo para proyectos críticos por tiempo, permitiendo que el equipo de software valore su proyecto sobre una base sólida. La prueba de humo comprende las siguientes actividades:

1. Los componentes software que han sido traducidos al código se integran en una "construcción". Una construcción incluye fichas de datos, librerías, módulos reutilizables y componentes de ingeniería que se requieren para implementar una o más funciones del producto.
2. Se diseña una serie de pruebas para descubrir errores que impiden a la construcción realizar su función adecuadamente. El objetivo será descubrir errores "bloqueantes" que tengan la mayor probabilidad de impedir al proyecto de software el cumplimiento de su planificación.
3. Es habitual en la prueba de humo que la construcción se integre con otras construcciones y que se aplica una prueba de humo al producto completo. La integración puede hacerse bien de forma descendente o ascendente.

La prueba de humo facilita una serie de beneficios cuando se aplica sobre proyectos de ingeniería de software complejos y críticos por su duración:

- Se minimiza los riesgos de integración.
- Se perfecciona la calidad del producto final.
- Se simplifican el diagnóstico y la corrección de errores.
- El progreso es fácil de observar.

3.3.5 Prueba de validación

La validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente. La validación del software se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos.

Se llevan a cabo dos pruebas:

Prueba Alfa	Prueba Beta
Se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de uso. Las pruebas Alfa se llevan a cabo en un entorno controlado.	Se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. El desarrollador no está presente en esta prueba. La prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que encuentran durante la prueba e informa al desarrollador. Como resultado de los problemas informados durante la prueba beta, el desarrollador del software lleva a cabo modificaciones y así prepara una versión del producto de software para toda la clase de clientes.

3.3.6 Prueba del sistema

Su propósito primordial es ejercitar profundamente el sistema, verificando que se hayan integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

Comprende las siguientes pruebas:

Prueba de recuperación
Esta prueba fuerza el fallo del software de muchas formas y verifica que la

recuperación se lleva a cabo apropiadamente.

Prueba de seguridad

Intenta verificar que los mecanismos de protección incorporadas en el sistema lo protegerán, de hecho, de accesos impropios.

Prueba de resistencia (stress)

Ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales. Por ejemplo:

1. Diseñar pruebas especiales que generen 10 interrupciones por segundo, cuando las normales son una o dos;
2. Incrementar las frecuencias de datos de entrada en un orden de magnitud con el fin de comprobar cómo responden las funciones de entrada.
3. Ejecutar casos de prueba que requieran el máximo de memoria o de otros recursos.
4. Diseñar casos de prueba que puedan dar problemas en un sistema operativo virtual.
5. Diseñar casos de prueba que produzcan excesivas búsquedas de datos residentes en disco.

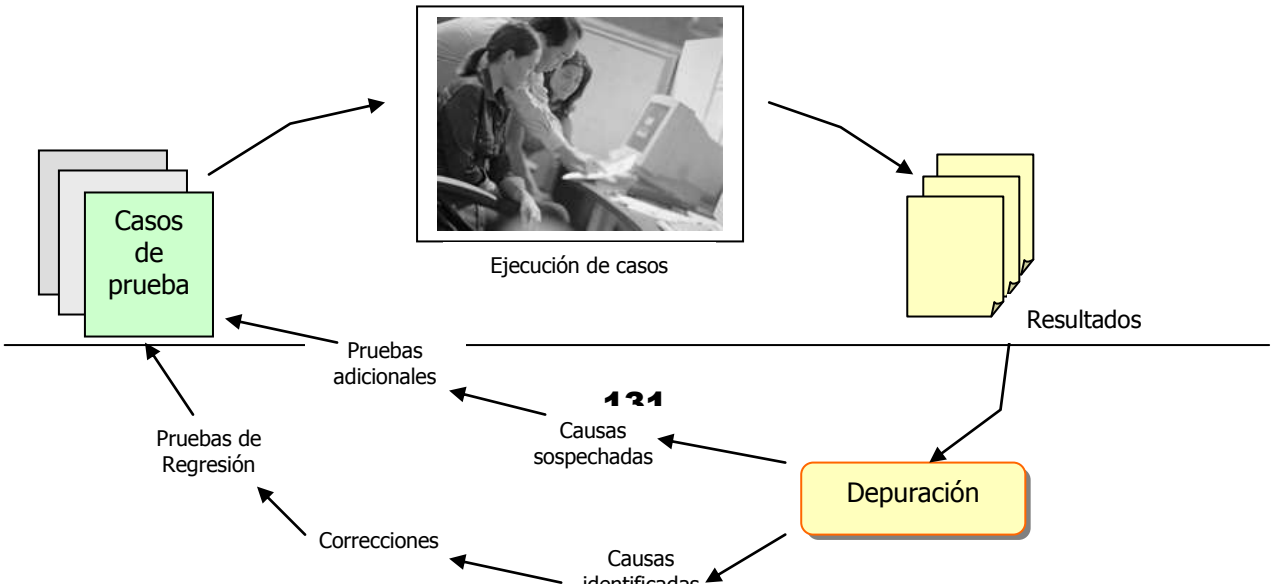
Esencialmente, el responsable de la prueba intenta romper el programa.

Prueba de rendimiento

Permite probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. La prueba de rendimiento se da durante todos los pasos del proceso de la prueba. Para llevar a cabo esta prueba, deben estar integrados completamente todos elementos del sistema.

3.3.7 Depuración

La depuración ocurre como consecuencia de una prueba efectiva. Cuando un caso de prueba descubre un error, la depuración es el proceso que provoca la eliminación del error.



El proceso de depuración comienza con la ejecución de un caso de prueba. Se evalúan los resultados y aparece una falta de correspondencia entre los esperados y los encontrados realmente. El proceso de depuración intenta hacer corresponder el sistema con una causa, llevando así a la corrección del error.

El proceso de depuración siempre tiene uno de los dos resultados siguientes:

1. Se encuentra la causa, se corrige y se elimina.
2. No se encuentra la causa.

En este último caso, la persona que realiza la depuración debe sospechar la causa, diseñar un caso de prueba que ayude a confirmar sus sospechas y el trabajo vuelve hacia atrás a la corrección del error de una forma interactiva.

Durante la depuración se encuentran errores que van desde lo ligeramente inesperado hasta lo catastrófico.

La depuración tiene el objetivo primordial de encontrar y corregir la causa de un error en el software.

3.4 MÉTRICAS TÉCNICAS DEL SOFTWARE

Las métricas técnicas para el software proporcionan una manera sistemática de valorar la calidad basándose en un conjunto de reglas. También proporcionan al ingeniero del software descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos.

3.4.1 Factores de calidad

Factores de calidad de McCall

McCall y Cavano definieron un juego de factores de calidad como los primeros pasos hacia el desarrollo de métricas de la calidad del software. Estos factores evalúan el software desde tres puntos de vista distintos:

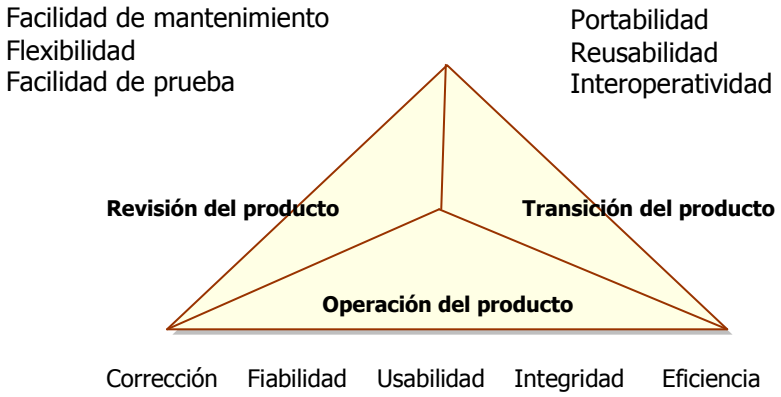
1. Operación del producto
2. Revisión del producto
3. Transición del producto

1. Operaciones del producto - Características operativas	
Corrección ¿Hace lo que se le pide?	El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente.
Fiabilidad ¿Lo hace de forma fiable todo el tiempo?	El grado que se puede esperar de una aplicación lleve a cabo las operaciones especificadas y con la precisión requerida.
Eficiencia ¿Qué recursos hardware y software necesito?	La cantidad de recursos hardware y software que necesita una aplicación para realizar las operaciones con los tiempos de respuesta adecuados.
Integridad ¿Puedo controlar su uso?	El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado.
Facilidad de uso ¿Es fácil y cómodo de manejar?	El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados

2. Revisión del producto - Capacidad para soportar cambios	
Facilidad de mantenimiento ¿Puedo localizar los fallos?	El esfuerzo requerido para localizar y reparar errores.
Flexibilidad ¿Puedo añadir nuevas opciones?	El esfuerzo requerido para modificar una aplicación en funcionamiento.
Facilidad de prueba ¿Puedo probar todas las opciones?	El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos.

3. Transición del producto - Adaptabilidad a nuevos entornos	
Portabilidad ¿Podré usarlo en otra máquina?	El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo.
Reusabilidad	Grado en que partes de una aplicación pueden

¿Podré utilizar alguna parte del software en otra aplicación?	utilizarse en otras aplicaciones
Interoperabilidad ¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?	El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos



Factores de calidad de McCall (Fuente: Roger Pressman, 2005)

Métrica para el esquema de puntuación

Las métricas pueden ir en forma de lista de comprobación para evaluar y puntuar atributos específicos del software.
McCall, propuso un esquema de puntuación en una escala del 0 (bajo) al 10 (alto). Se emplean las siguientes métricas en el esquema de puntuación:

Facilidad de auditoría	La facilidad con la que se puede comprobar el cumplimiento de los estándares.
Exactitud	La exactitud de los cálculos y del control.
Estandarización de comunicaciones	El grado de empleo de estándares de interfaces, protocolos y anchos de banda.
Complexión	El grado con que se ha logrado la implementación total de una función.
Concisión	Lo compacto que es el programa en términos de líneas de código.
Consistencia	El empleo de un diseño uniforme y de técnicas de documentación a lo largo del proyecto de desarrollo del software
Estandarización de datos	El empleo de estructuras y tipos de datos estándares a lo largo del programa.
Tolerancia al error	El daño causado cuando un programa encuentra un error.
Eficiencia de ejecución	El rendimiento del funcionamiento de un programa.
Capacidad de expansión	El grado con que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Generalidad	La amplitud de aplicación potencial de los componentes del programa.
Independencia del hardware	El grado con que se desacopla el software del hardware donde opera.
Instrumentación	El grado con que el programa vigila su propio funcionamiento e

	identifica los errores que ocurren.
Modularidad	La independencia funcional de componentes de programa.
Operatividad	La facilidad de operación de un programa
Seguridad	La disponibilidad de mecanismos que controlan o protegen los programas y los datos.
Autodocumentación	El grado en que el código fuente proporcionan documentación significativa
Simplicidad	El grado de facilidad con que se puede entender un programa.
Independencia del sistema software	El grado de independencia de programa respecto a las características del lenguaje de programación no estándar, características del sistema operativo y otras restricciones del entorno.
Trazabilidad	La capacidad de seguir una representación del diseño o un componente real del programa hasta los requisitos.
Formación	El grado en que ayuda el software a manejar el sistema o los nuevos usuarios.

A continuación, se presenta la relación entre los factores de calidad del software y las métricas de la lista anterior.

Métrica de la calidad del software / **Factor de calidad**	Corrección	Fiabilidad	Eficiencia	Integridad	Mantenimiento	Flexibilidad	Capacidad de pruebas	Portabilidad	Reusabilidad	Interoperatividad	Usabilidad
Facilidad de auditoria				X			X				
Exactitud		X									
Estandarización de comunicaciones										X	
Complexión	X										
Complejidad		X				X	X				
Concisión			X		X	X					
Consistencia	X	X			X	X					
Estandarización de datos										X	
Tolerancia a errores		X									
Eficiencia de ejecución			X								
Capacidad de expansión						X					
Generalidad						X		X	X	X	
Independencia del hardware								X	X		
Instrumentación				X	X		X				
Modularidad		X			X	X	X	X	X	X	

Operatividad			X								X
Seguridad				X							
Autodocumentación					X	X	X	X	X		
Simplicidad		X			X	X	X				
Independencia del sistema								X	X		
Trazabilidad	X										
Facilidad de formación											X

3.4.2 FURPS

El modelo de McCall ha servido de base para modelos de calidad posteriores, y este es el caso del modelo FURPS, producto del desarrollo de Hewlett-Packard. En este modelo se desarrollan un conjunto de factores de calidad de software, bajo el acrónimo de FURPS.

F	Functionality - funcionalidad
U	Usability – usabilidad – facilidad de uso
R	Reliability - confiabilidad
P	Performance - desempeño
S	Supportability - capacidad de soporte

La siguiente tabla, presenta la clasificación de los atributos de calidad que se incluyen en el modelo, junto con las características asociadas a cada uno (Pressman, 2002).

Factor de Calidad	Atributos
Funcionalidad	• Características y capacidades del programa • Generalidad de las funciones • Seguridad del sistema
Facilidad de uso	• Factores humanos • Factores estéticos • Consistencia de la interfaz • Documentación
Confiabilidad	• Frecuencia y severidad de las fallas • Exactitud de las salidas • Tiempo medio de fallos • Capacidad de recuperación ante fallas • Capacidad de predicción
Rendimiento	• Velocidad del procesamiento • Tiempo de respuesta • Consumo de recursos • Rendimiento efectivo total

	• Eficacia
Capacidad de Soporte	• Extensibilidad • Adaptabilidad • Capacidad de pruebas • Capacidad de configuración • Compatibilidad • Requisitos de instalación

El modelo FURPS incluye, además de los factores de calidad y los atributos, restricciones de diseño y requerimientos de implementación, físicos y de interfaz. Las restricciones de diseño especifican o restringen el diseño del sistema. Los requerimientos de implementación especifican o restringen la codificación o construcción de un sistema (por ejemplo, estándares requeridos, lenguajes, políticas). Por su parte, los requerimientos de interfaz especifican el comportamiento de los elementos externos con los que el sistema debe interactuar. Por último, los requerimientos físicos especifican ciertas propiedades que el sistema debe poseer, en términos de materiales, forma, peso, tamaño (por ejemplo, requisitos de hardware, configuración de red).

Factores de calidad ISO 9126

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software (Pressman, 2002). Este estándar es una simplificación del Modelo de McCall, e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. El estándar provee una descomposición de las características en subcaracterísticas, que se muestran en la siguiente tabla:

Característica	Subcaracterística
Funcionalidad	• Adecuación • Exactitud • Interoperabilidad • Seguridad
Confiabilidad	• Madurez • Tolerancia a fallas • Recuperabilidad
Usabilidad	• Entendibilidad • Capacidad de aprendizaje • Operabilidad
Eficiencia	• Comportamiento en tiempo • Comportamiento de recursos
Mantenibilidad	• Analizabilidad

	• Modificabilidad • Estabilidad • Capacidad de pruebas
Portabilidad	• Adaptabilidad • Instalabilidad • Reemplazabilidad

ISO/IEC 9126 no son necesariamente usados para mediciones directas (Pressman, 2002), pero proveen una valiosa base para medidas indirectas, y una excelente lista para determinar la calidad de un sistema.

Es importante establecer una estructura fundamental y un conjunto de principios básicos para la medición de métricas técnicas para el software.

Los principios básicos de la medición, sugeridos por Roche, pueden caracterizarse mediante cinco actividades:

Formulación	Obtención de medidas y métricas del software apropiadas para la representación del software.
Colección	Mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
Análisis	Cálculo de las métricas y aplicación de herramientas matemáticas.
Interpretación	Evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
Realimentación	Recomendaciones obtenidas de la interpretación de métricas técnicas transmitidas al equipo software.

Los principios que se pueden asociar con la formulación de las métricas técnicas son los siguientes:

- Los objetivos de la medición que deben establecerse antes de empezar la recolección de datos.
- Todas las técnicas sobre métricas deben definirse sin ambigüedades.
- Las métricas deberían obtenerse basándose en una teoría válida para el dominio de aplicación.
- Hay que hacer las métricas a medida para acomodar mejor los productos y procesos específicos.

Roche sugiere los siguientes principios para la recolección y análisis de datos:

- Siempre que sea posible, la recogida de datos y el análisis debe automatizarse.

- Se deben aplicar técnicas estadísticas válidas para establecer las relaciones entre los atributos internos del producto y las características externas de la calidad.
- Se deben establecer directrices de interpretación y recomendaciones para todas las métricas.

La métrica obtenida y las medidas que conducen a ello deben tener las siguientes características:

- Simples y fáciles de calcular.
- Empírica e intuitivamente persuasivas.
- Consistentes y objetivas.
- Consistentes en el empleo de unidades y tamaños.
- Independiente del lenguaje de programación.
- Un mecanismo eficaz para la realimentación de calidad.

3.5 Métricas del modelo del software

3.5.1 Métricas del modelo de análisis

En esta fase, las métricas técnicas proporcionan una visión interna a la calidad del modelo de análisis. Estas métricas examinan el modelo de análisis con la intención de predecir el "tamaño" del sistema resultante; es probable que el tamaño y la complejidad del diseño estén directamente relacionados.

Dentro de las métricas del modelo de análisis tenemos:

Métricas basadas en la Función

La métrica del punto de función se utiliza como medio para predecir el tamaño de un sistema obtenido a partir de un modelo de análisis. Para visualizar esta métrica se utiliza un diagrama de flujo de datos, el cual se evaluar para determinar las siguientes medidas clave que son necesarias para el cálculo de la métrica de punto de función:

- Número de entradas del usuario
- Número de salidas del usuario
- Número de consultas del usuario
- Número de archivos
- Número de interfaces externas

La conceptualización de esta métrica ya fue expuesta en la unidad 2, de éste módulo.

Métrica Bang

Puede aplicarse para desarrollar una indicación del tamaño del software a implementar como consecuencia del modelo del análisis.
Para calcular la métrica bang, el desarrollador de software evalúa primero un conjunto de primitivas. Las primitivas se determinan evaluando el modelo de análisis y desarrollando cuentas para los siguientes elementos:

Primitivas funcionales (Pfu)	Transformaciones que aparecen en el nivel inferior de un diagrama de flujo de datos.
Elementos de datos (ED)	Los atributos de un objeto de datos, los elementos de datos no compuestos y aparecen en el diccionario de datos.
Objetos (OB)	Objetos de datos.
Relaciones (RE)	Las conexiones entre objetos de datos.
Estados (ES)	El número de estados observables por el usuario en el diagrama de transición de estados.
Transiciones (TR	El número de transacciones de estado en el diagrama de transición de estado.

Además, se determinan medidas adicionales para:

Primitivas modificadas de función manual (PMFu)	Funciones que caen fuera del límite del sistema y que deben modificarse para acomodarse al nuevo sistema.
Elementos de datos de entrada (EDE)	Aquellos elementos de datos que se introducen en el sistema.
Elementos de datos de salida (EDS)	Aquellos elementos de datos que se sacan en el sistema.
Elementos de datos retenidos (EDR)	Aquellos elementos de datos que son retenidos (almacenados) por el sistema.
Muestras (tokens) de datos (TCi)	Las muestras de datos que existen en el límite de la i-ésima primitiva funcional (evaluada para cada primitiva).
Conexiones de relación (Rei)	Las relaciones que conectan el i-ésimo objeto en el modelo de datos con otros objetos.

3.5.2 Métricas del modelo de diseño

Se concentran en las características de la arquitectura del programa, con énfasis en la estructura arquitectónica y en la eficiencia de los módulos.

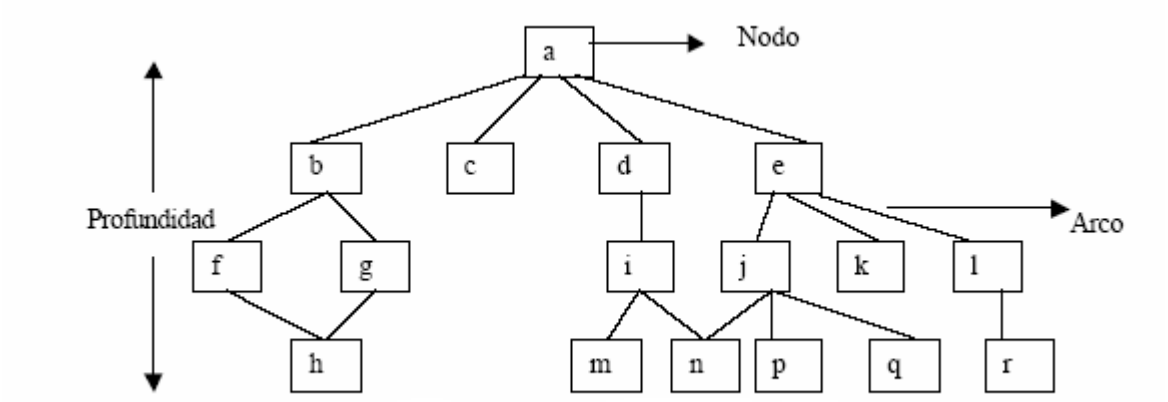
Estas métricas son de caja negra en el sentido que no requieren ningún conocimiento del trabajo interno de un módulo en particular del sistema.
Card y Glass definen las siguientes tres medidas de complejidad:

La complejidad estructural, S(i), de un módulo i se define de la siguiente manera: $S(i)=f^2_{out}(i)$. Donde $f_{out}(i)$ es la expansión del módulo i. La expansión indica el número de módulos que son invocados directamente por el módulo i.

La complejidad del sistema, C(i), se define como la suma de las complejidades estructural y de datos : C(i)= S(i) + D(i)

La complejidad de datos, D(i), proporciona una indicación de la complejidad en la interfaz interna de un módulo i y se define como: $D(i)=v(i)/[f_{out}(i) + 1]$. Donde v(i) es el número de variables de entrada y salida que entran y salen del módulo i.

Fenton sugiere varias métricas de morfología simples que permiten comparar diferentes arquitecturas mediante un conjunto de dimensiones directas.



Fuente: Roger S.Pressman, 2002

Las métricas a aplicar son:

- Tamaño= n + a. Donde n es el número de nodos (módulos) y a es el número de arcos (líneas de control). Para la arquitectura mostrada se tiene tamaño= 17+18=35.
- Profundidad= camino más largo desde el nodo raíz a un nodo hoja. Para el ejemplo Profundidad= 4
- Amplitud=número máximo de nodos de cualquier nivel de la arquitectura. Para el ejemplo amplitud= 6
- Relación arco a nodo, r=a/n, mide la densidad de conectividad de la arquitectura y proporciona una indicación sencilla de acoplamiento de la arquitectura. Para el ejemplo r=18/17= 1.06

3.5.3 Métricas del código fuente

Utiliza un conjunto de medidas primitivas que pueden obtenerse una vez que se han generado o estimado el código después de completar el diseño.

Estas medidas son:

n_1 : número de operadores diferentes que aparecen en el programa.
n_2 : número de operandos diferentes que aparecen en el programa.
N_1 : número total de veces que aparece el operador.
N_2 : número total de veces que aparecen el operando.

Halstead utiliza medidas primitivas para desarrollar expresiones par la longitud global del programa; volumen mínimo potencial para un algoritmo; el volumen real (número de bits requeridos para especificar un programa); el nivel del programa (una medida de la complejidad del software); nivel del lenguaje (una constante para un lenguaje dado); y otras características tales como el esfuerzo de desarrollo, tiempo de desarrollo e incluso el número esperado de fallos en el software.

Halstead propone las siguientes métricas:

- Longitud N se puede estimar como: $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$.
- Volumen de programa se define como: $V = N\, n_1 \log_2(n_1 + n_2)$.

  Tomando en cuenta que V variará con el lenguaje de programación y representa el volumen de información (en bits) necesarios para especificar un programa.

3.5.4 Métricas para pruebas

Las métricas para pruebas se concentran en el proceso de prueba, no en las características técnicas de las pruebas mismas. En general, los responsables de las pruebas deben fiarse en las métricas de análisis, diseño y código para que sirvan de guía en el diseño y ejecución de los casos de prueba.

El esfuerzo de las pruebas también se puede estimar utilizando métricas obtenidas de las medidas de Halstead.

Usando la definición del volumen de un programa, V, y nivel de programa, NP, el esfuerzo de la ciencia del software puede calcularse como:

$$NP = 1/[(n1/2) \times (N2/n2)] \quad \text{(Ec. 1)}$$
$$e = V/NP \quad \text{(Ec. 2)}$$

El porcentaje del esfuerzo global de pruebas a asignar a un módulo k se puede estimar utilizando la siguiente relación:

$$\text{Porcentaje de esfuerzo de pruebas}(k) = e(k)/\sum e(i) \quad \text{(Ec. 3)}$$

Donde e(k) se calcula para el módulo k utilizando las ecuaciones (Ec. 1) y (Ec. 2), la suma en el denominador de la ecuación (Ec. 3) es la suma del esfuerzo de la ciencia del software a lo largo de todos los módulos del sistema.

A medida que se van haciendo las pruebas, tres medidas diferentes proporcionan una indicación de la compleción de las pruebas:

Medida de amplitud de las pruebas.	Proporciona una indicación de cuantos requisitos se han probado del número total de ellos. Indica la compleción del plan de pruebas.
Profundidad de las pruebas	Medida del porcentaje de los caminos básicos independientes probados con relación al número total de estos caminos en el programa. Se puede calcular una estimación razonablemente exacta del número de caminos básicos sumando la complejidad ciclomática de todos los módulos del programa.
Perfiles de fallos	Se emplean para dar prioridad y categorizar los errores. La prioridad indica la severidad del problema. Las categorías de los fallos proporcionan una descripción de un error, de manera que se puedan llevar a cabo análisis estadístico de errores.

3.5.5 Métricas del mantenimiento

Todas las métricas descritas pueden utilizarse para el desarrollo de nuevo software y para el mantenimiento del existente.

El estándar IEEE 982.1-1988 sugiere el índice de madurez del software (IMS) que proporciona una indicación de la estabilidad de un producto software basada en los cambios que ocurren con cada versión del producto. Con el IMS se determina la siguiente información:

El índice de madurez del software se calcula de la siguiente manera:

- MT= Número de módulos en la versión
- actualFc = Número de módulos en la versión actual que se han cambiado
- Fa= Número de módulos en la versión actual que se han añadido
- Fe= Número de módulos en la versión actual que se han eliminado

A medida que el IMS se aproxima a 1 el producto se empieza a estabilizar. El IMS puede emplearse también como métrica para la planificación de las actividades de mantenimiento del

IMS= [MT - (Fc + Fa + Fe)]/MT

software.

ACTIVIDADES COMPLEMENTARIAS

1. Con sus propias palabras, describa las diferencias entre **verificación** y **validación**.

2. Haga una lista de algunos problemas que puedan estar asociados con la creación de un grupo independiente de prueba. ¿Están formados por las mismas personas el GIP y el grupo SQA?

3. ¿Quién debe llevar a cabo la prueba de validación -el desarrollador del software o el usuario-? Justifique su respuesta.

RECURSOS DE SOFTWARE LIBRE PARA INGENIERÍA DEL SOFTWARE

Las siguientes herramientas han sido compiladas por Tigris.org, página web http://www.tigris.org/

Tigris.org es una comunidad, de tamaño medio, de código abierto enfocado en la construcción de mejores herramientas para el desarrollo colaborativo de software. La misiónd e Tigris.org es construir Herramientas de Ingeniería de Software Open Source o de código abierto. Tigris.org está alojado por CollabNet, se convierte en el mayor y más completo movimiento de código abierto que ha más atraído desarrolladores de código abierto de muchas organizaciones.

CATEGORÍA: scm - Software configuration management tools

Enlace: http://scm.tigris.org/

Comúnmente conocida como control de versiones, y se logra utilizando herramientas como CVS. SCM es la clave para cualquier proyecto de desarrollo de software, que desee actualizar, mejorar o apoyar un nuevo desarrollo con base en uno anterior logrado.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
Subversion	Subversion es una versión de código abierto de un sistema de control. Fundada en 2000 por CollabNet, Inc., el proyecto y software Subversion han tenido un éxito increíble en la última década. Subversion ha gozado y sigue gozando de amplia adopción, tanto en el campo de código abierto como en el mundo empresarial. Subversion existe para ser universalmente reconocido y adoptado como un código abierto, sistema centralizado de control de versión, el cual se caracteriza por su fiabilidad como un refugio seguro para los datos valiosos, la simplicidad de su modelo y su uso, y su capacidad para apoyar las necesidades de una amplia variedad de usuarios y proyectos, desde particulares a las operaciones empresariales a gran escala.	http://subversion.apache.org/
Subclipse	Subclipse es un eqipo proveedor de soporte para el plug-in Eclipse, para la subversión dentro de la IDE de Eclipse. El	http://subclipse.tigris.org/

	software se distribuye bajo la Licencia Pública Eclipse (EPL) 1.0 Licencia de código abierto. Este software básicamente provee herramientas gráficas para la organización de la documentación de cualquier proyecto de software, puede llevarse al más mínimo nivel de descripción detalles del software como: modificaciones, instalaciones, defectos, correcciones, soportes, etc.	
TortoiseSVN	TortoiseSVN es una herramienta muy fácil de usar para el control de revisiones / control de versiones / software de control de código fuente para Windows. No es una integración de un IDE específico, por lo que se puede utilizar con herramientas de desarrollo de cualquier tipo, especialmente propietarias. TortoiseSVN es de uso gratuito	http://tortoisesvn.tigris.org/
RapidSVN	RapidSVN, es un software con herramientas visuales especiales y fáciles de manejar para clientes, que les permite un mejor control de revisiones de arquitecturas.	http://rapidsvn.tigris.org/

CATEGORÍA: issuetrack- Defect and issue tracking tools

Enlace: http://issuetrack.tigris.org/

Herramientas de seguimiento a problemas y defectos del software. Todo el software presenta defectos durante el desarrollo, y muchas veces en sus versiones iniciales. Los desarrolladores de software deben realizar un seguimiento de los miles de defectos y solicitudes de cambios que se producen en el transcurso del proceso de desarrollo. Hacer un seguimiento de esta información en un archivo de hoja de cálculo o una bandeja de entrada de bugs.txt o de correo electrónico no es suficiente debido a la complejidad de la tarea de seguimiento de incidencias y su importancia para todas las fases de desarrollo.

Seguimiento de problemas es una generalización de seguimiento de defectos. Los temas incluyen todas las peticiones de características y mejoras, así como las tareas de desarrollo de módulos o piezas del software ("gastos generales" el trabajo que hay que hacer). Seguimiento de incidencias también se puede utilizar para la atención al cliente después de que el producto es enviado.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
scarab	Artefacto como sistema de seguimiento.	http://scarab.tigris.org/
bbapi	API de Java para tableros de anuncios	http://bbapi.tigris.org/
leo	Editor de lectura y escritura con esquemas	http://leo.tigris.org/

phrac	Wiki escrito en PHP5 y sistema de seguimiento de fallos para proyectos de software	http://phrac.tigris.org/

CATEGORÍA: Requirements - Software requirements management tools

Enlace: http://requirements.tigris.org/

Esta categoría hace referencia a herramientas para administración de requerimientos de software. Requerimientos adecuados y actualizados son la clave para un desarrollo de software efectivo. Esta categoría incluye proyectos de herramientas que ayudan a expresar, gestionar, validar, actualizar o formalizar los requisitos para desarrollo. Además, esta categoría incluye proyectos que organizan contenidos, ya que describe los métodos y modelos para los requisitos de software.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
xmlbasedsrs	Especificaciones de requerimientos para desarrollo de software como documentos XML	http://xmlbasedsrs.tigris.org/
ipmrt	Herramientas para gestión de requisitos de proyectos inteligentes	http://ipmrt.tigris.org/
jreq	Sistema para administración de proyectos flexibles	http://jreq.tigris.org/
reqs	Herramientas y lenguajes sencillos para ordenamiento, filtrado y procesamiento de información de proyectos software.	http://reqs.tigris.org/

CATEGORÍA: Design - Software design tools

Enlace: http://design.tigris.org/

Herramientas para apoyar el diseño del software. El diseño de software es el punto desde los requerimientos hacia la implementación. Las decisiones de diseño (incluyendo la arquitectura) son las decisiones orientadas a la solución de más alto nivel tomadas en el proceso de desarrollo de software. Como tales, tienen numerosas implicaciones en las etapas que la siguen.

Esta categoría alberga herramientas que ayudan a los desarrolladores de software para diseño. que organizan contenidos, ya que describe los métodos y modelos para los requisitos de software.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
argouml	Una herramienta de diseño UML con apoyo cognitivo	http://argouml.tigris.org/
argoeclipse	PlugIn Eclipse para diseño UML usando ArgoUML	http://argoeclipse.tigris.org/
argoprint	Permite la generación de vista de impresión de los modelos de ArgoUML	http://argoprint.tigris.org/
argosoffice	Plugin Star/OpenOffice para ArgoUML	http://argosoffice.tigris.org/

CATEGORÍA: Techcomm- Tools for technical communications

Enlace: http://techcomm.tigris.org/

Herramientas para comunicaciones técnicas. Las comunicaciones técnicas suceden en cada etapa del proceso de desarrollo de software. De hecho, muchas teorías de la gestión de proyectos comienzan con la suposición de que la comunicación es el mayor cuello de botella del proceso y por lo tanto todo lo demás debe ser planificado alrededor de ella.

Los proyectos en esta categoría tratan de ayudar a los desarrolladores de comunicar aspectos técnicos. Tecnologías relevantes incluyen grupos de correo electrónico, noticias, foros web, documentos estructurados, herramientas para la toma de decisiones en grupo y la justificación del diseño.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
subetha	Moderno y sofisticado gestor de listas de correo	http://subetha.tigris.org/
eyebrowse	Navegador de archivos de lista de correos	http://eyebrowse.tigris.org/
productreviews	Revisiones de productos creados basados en criterios orientados al usuario.	http://productreviews.tigris.org/
cowiki	Herramientas de colaboración web orientadas a objetos GPLed escritas en PHP5	http://cowiki.tigris.org/

CATEGORÍA: Construction - Coding, testing, and debugging tools

Enlace: http://construction.tigris.org/

Estas herramientas son usadas durante la fase de implementación.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:

antelope	Interfaz gráfica de usuario para Ant	http://antelope.tigris.org/
frameworx	J2EE framework, mecanismo de construcción JAVA	http://frameworx.tigris.org/
build-interceptor	Intercepta archivos .i de un proyecto mientras este se construye con gcc.	http://build-interceptor.tigris.org/
propel	Servicio de búsqueda persistente de objetos para PHP5	http://propel.tigris.org/

CATEGORÍA: testing- Software testing automation tools

Enlace: http://testing.tigris.org/

Herramientas para automatización de las pruebas de software. La creación de un sistema de software es una tarea increíblemente compleja y propensa a errores. Al igual que con cualquier actividad de ingeniería, los defectos surgen en todas las fases de desarrollo y deben ser encontrados y resueltos antes de que impacten negativamente en los usuarios.

Construir la calidad en un producto es la mejor manera de lograr un resultado de alta calidad. Eso significa escuchar atentamente a los usuarios, entendiendo sus necesidades, requisitos precisos por escrito, elección de herramientas de alta calidad y componentes reutilizables y la realización de pruebas continuas. El logro de un producto de alta calidad también exige probar su calidad en cada etapa. En el largo plazo, la calidad también depende del proceso de desarrollo de software, y la capacidad de la organización de desarrollo para mejorar su proceso en lugar de simplemente encontrar y reparar defectos individuales.

Con frecuencia en la industria de software, la prueba se realiza manualmente y de manera incompleta. Esto permite que los defectos no se detecten y le quita esfuerzo al equipo de desarrollo que podría ser mejor aprovechado en otras actividades.

Esta categoría alberga los proyectos que buscan la construcción de herramientas para pruebas de software.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
aut	Para pruebas unitarias avanzadas	http://aut.tigris.org/
maxq	Es una herramienta web para testeos funcionales	http://maxq.tigris.org/
perfbase	Administración y análisis de los resultados de experimientos y pruebas de software	http://perfbase.tigris.org/
storyteller	Pruebas de aceptación automática para. Net	http://storyteller.tigris.org/

CATEGORÍA: deployment- Tools for deploying and updating software

Enlace: http://deployment.tigris.org//

Herramientas para implementación y actualización de software. Herramientas de implementación ayudan a automatizar y gestionar la tarea de entregar versiones empaquetadas de software para los usuarios finales.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
autoinst	Sistema controlador de versiones para host UNIX.	http://deployment.tigris.org/
carnarvon	Herramienta para análisis arqueológico de software.	http://carnarvon.tigris.org/
current	Herramientas para administración e implementación de paquetes open-source	http://current.tigris.org/
vordos	Plataforma de desarrollo de aplicaciones en internet	http://vordos.tigris.org/

CATEGORÍA: process- Projects related to software development processes

Enlace: http://process.tigris.org/

Proyectos relacionados a procesos de desarrollo de software. El proceso de movimiento de mejora de software realmente se fue arraigando en la década de 1990. Se centró la atención de la ingeniería de software en el proceso de desarrollo en lugar de en las herramientas o tecnologías específicas. Esta categoría tiene proyectos que definen y apoyan procesos de desarrollo de software.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
dig	Herramientas para la excavación de código antiguo.	http://dig.tigris.org/
lptools	Una suite de literatura en programación con una herramienta de construcción programable.	http://lptools.tigris.org/
nteam	Plataforma de colaboración para equipos de desarrollo.	http://nteam.tigris.org/
ReadySET	Plantillas de ingeniería del software listas para usar	http://readyset.tigris.org/

CATEGORÍA: libraries- Reusable software components

Enlace: http://libraries.tigris.org/

Componentes de software reutilizables. Los sistemas actuales de software son cada vez más complejos. Sin embargo, el tiempo y esfuerzo disponible para producir estos sistemas son cada vez más cortos.

La reutilización del software es una de las soluciones más importantes para el desafío de construir sistemas complejos. Una reutilización efectiva requiere:

- Repositorios de fácil acceso a componentes de alta calidad
- Estándares en tecnología, documentación y procesos de software
- Herramientas para encontrar, entender, evaluar, adaptar e integrar los componentes reutilizables
- Apoyo efectivo de la comunidad de desarrolladores que está reutilizando cada biblioteca o librería

Esta categoría en tigris.org alberga proyectos de código abierto que están produciendo componentes de software reutilizables.

Dentro de esta categoría se pueden mencionar las siguientes herramientas:

Herramienta	Uso	Disponible en:
gef	Framework para edición gráfica en Java	http://gef.tigris.org/
axion	Base de datos JAVA	http://axion.tigris.org/
style	CSS para aplicaciones web	http://style.tigris.org/
SSTree	Arbol Super Simple Java Script	http://sstree.tigris.org/

RECURSOS BIBLIOTECA VIRTUAL UNAD

Acceso a Biblioteca Virtual Unad

Consultar la Biblioteca Virtual de la Unad es muy fácil. Pasos a seguir-->

1. Ingresar a la página web de la Unad: http://www.unad.edu.co/home/

2. Allí seleccionar la opción "Estudiantes" en el menú de la parte superior.

3. Dentro de estudiantes ubicar la opción Servicios y allí dar click en la opción Biblioteca. O ingresar al enlace directo: http://www.unad.edu.co/biblioteca/

Una vez en la página de Biblioteca, se encontrará toda la información de buscadores, accesos, ayudas y apoyo para consulta de texto, libros y demás material bibliográfico disponible.

Para consulta Ingeniería del Software

Material de Biblioteca Virtual Unad para consulta y refuerzo de temáticas del curso 301404-Ingeniería del Software

Recurso:	elibro
Título:	Los niveles de servicio en la ingeniería del software
Autor:	La Red Martínez, David Luis
Autor Adic./ Editor:	Peláez Sánchez, José Ignacio
Año:	2006
Materia:	Ingeniería de programas y sistemas de programación. Software engineering. Ingenieria de programas y sistemas de programacion
Impresor:	El Cid Editor : , , Argentina
Tipo:	book
Idioma:	es
Ver Texto Completo:	http://site.ebrary.com/lib/unadsp/docDetail.action?adv.x=1& amp;d=all& amp;f00=all& amp;f01=& amp;f02=& amp;hitsPerPage=500& amp;p00=Ingenier%C3%ADa+del+software& amp;p01=& amp;p02=& amp;page=1& amp;id=10121909

Recurso:	elibro
Título:	Planificación y gestión de proyectos informáticos
Autor:	Gutiérrez de Mesa, José Antonio
Autor Adic./ Editor:	Pagés Arévalo, Carmen
Año:	2008
Materia:	Informática. -- unescot Gestión. -- unescot Management information systems. -- lcsh Libros electronicos. -- local
ISBN:	9788481387940
Impresor:	Servicio de Publicaciones. Universidad de Alcalá : , , España
Idioma:	es
Ver Texto Completo:	http://site.ebrary.com/lib/unadsp/docDetail.action?adv.x=1& amp;d=all& amp;f00=all& amp;f01=& amp;f02=& amp;hitsPerPage=500& amp;p00=Ingenier%C3%ADa+del+software& amp;p01=& amp;p02=& amp;page=1& amp;id=10280334

Recurso:	ebrary
Título:	Metrics for Software Conceptual Models
Autor:	Genero, Marcela
Autor Adic./	Piattini, Mario
Editor:	Calero, Coral
Año:	2005
Materia:	Computer software -- Mathematical models. Computers.
ISBN:	9781860946066
Impresor:	Imperial College Press : London, , GBR
Idioma:	en
Ver Texto Completo:	http://site.ebrary.com/lib/unad/docDetail.action?adv.x=1& amp;d=all& amp;f00=all& amp;f01=& amp;f02=& amp;hitsPerPage=500& amp;p00=Ingenier%C3%ADa+del+software& amp;p01=& amp;p02=& amp;page=1& amp;id=10091225

Recurso:	Gale Virtual Reference Library - ebook (Gale Group / InfoTrac)
Título:	Calidad de Software y Modelos de Madurez del Proceso
Cita:	Ingenieria de Software Mexico City: Cengage Learning, 2005
Año:	2005
Materia:	Technology
ISBN:	978-607-481-062-2
Fecha Publicación:	20050101
Tipo:	Topic Overview
Ver Texto Completo:	Electronic resource (PDF) Electronic resource (HTML)

Recurso:	Gale Virtual Reference Library - ebook (Gale Group / InfoTrac)
Título:	Modelos Recientes
Cita:	Ingenieria de Software Mexico City: Cengage Learning, 2005
Año:	2005
Materia:	Technology
ISBN:	978-607-481-062-2
Fecha Publicación:	20050101
Tipo:	Topic Overview
Ver Texto Completo:	Electronic resource (PDF) Electronic resource (HTML)

BIBLIOGRAFIA

IMPRESA

BRAUDE. Ingeniería de software, una perspectiva orientada a objetos. México. 2003. Alfaomega grupo editor. S.A.

GRUEGGE, BERND y DUTOIT, Allen H. Ingeniería de software orientado a objetos. México. 2002. Pearson Educación.

HUMPHREY, Watts S. Introducción al proceso de software personal. Pearson Addison wesley. 2001.

MEYER, Bertrand. Construcción de software orientado a objetos. Segunda edición. Madrid. 1999. Prentice Hall.

NORRIS. Ingeniería de software explicada. Grupo Noriega editores de Colombia.

PIATTINI, Mario. VILLALBA, Jose y otros. Mantenimiento del software: modelos, técnicas y métodos para la gestión del cambio. Editorial Alfaomega-Rama.

PRESSMAN, Roger S. Ingeniería del Software. Un enfoque práctico. Quinta edición. España. 2002. Editorial McGraw Hill.

PFLEEGER, Shari Lawrence. Ingeniería de software, teoría y práctica. 1ª. Edición. Buenos Aires. Pearson educación. 2002

SOMMERVILLE, Ian. Ingeniería de software. 6ª. Edición. Pearson Addison Wesley. 2001

ELECTRÓNICA

http://asqc.org
http://www.gestiopolis.com/recursos/documentos/fulldocs/eco/diagramapareto.htm
http://campus.fortunecity.com/defiant/114/iso9000.htm
http://www.iso.org
http://www.well.com/user/vision/sqa.html
http://www.processimprovement.com/resources/sqa.htm
http://www.softwaretestinginstitute.com/Profession.html