PROYECTOS DE DESARROLLO DE SOFTWARE (VERSIÓN 1.0)

Marco Villalobos Abarca

Universidad de Tarapacá
Facultad de Ingeniería
Departamento Ingeniería en Computación e Informática

Inscripción Registro de Propiedad Intelectual N° 303.577

Arica, Chile

2019

UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

DEPARTAMENTO INGENIERÍA EN COMPUTACIÓN E INFORMÁTICA

PROYECTOS DE DESARROLLO DE SOFTWARE

(Versión 1.0)

INGENIERÍA CIVIL EN COMPUTACIÓN E INFORMÁTICA

DR. MARCO VILLALOBOS ABARCA

ARICA – CHILE

2019

Textos y libros de Ingeniería de Software se encuentran en grandes cantidades en bibliotecas o la web y muchos de ellos en varias ediciones. Entonces, ¿por qué escribir uno más?. Simplemente porque no existe uno con orientación hacia proyectos y el contexto empresarial que le da sentido a la Ingeniería de Software.

El Autor.

CONTENIDO

INTRODUCCIÓN

PARTE I: CONTEXTO FORMATIVO

- **¿POR QUÉ PROYECTOS?**
- **PLANIFICACIÓN DEL CURSO**
- **PROCESO FORMATIVO**
- **PIEDRAS ANGULARES**

PARTE II: PROYECTOS DE DESARROLLO DE SOFTWARE

- **CAPÍTULO 1: SOFTWARE Y PROYECTO**
- **CAPÍTULO 2: INICIAR UN PROYECTO**
- **CAPÍTULO 3: PLANIFICAR UN PLROYECTO**
- **CAPÍTULO 4: EJECUTAR UN PROYECTO**
- **CAPÍTULO 5: CONTROLAR UN PROYECTO**
- **CAPÍTULO 6: CERRAR UN PROYECTO**

CONCLUSIONES

REFERENCIAS BIBLIOGRÁFICAS

ANEXOS - PROCEDIMIENTOS

GLOSARIO

PARTE I:
CONTEXTO FORMATIVO

¿POR QUÉ PROYECTOS?

Existe una brecha marcada entre lo que se enseña en el aula de clase y la realidad del desarrollo de software en las organizaciones. Esta situación ha motivado una reflexión en el mundo académico acerca de las competencias y habilidades que deben ser desplegadas por los futuros desarrolladores de software y las estrategias pedagógicas que pueden ser utilizadas de manera que sus experiencias de aprendizaje estén altamente influenciadas por las prácticas, técnicas y modos de trabajo que exige el desarrollo de software de calidad a escala industrial (Stiller y LeBland, 2002; Liu et. al., 2002; Bracken, 2003; Alsmadi y Abul-Huda, 2010). Por lo anterior, en la carrera de Ingeniería Civil en Computación e Informática se ha implementado una estrategia pedagógica basada en el Aprendizaje Basado en Proyectos (ABP).

Actualmente el ABP ha tenido una gran aceptación en la Educación Superior, sin embargo la abreviación ABP se ha utilizado tanto para el aprendizaje basado en problema como para el aprendizaje basado en proyecto. Kolmos et al. (2009), establecen que hay razones para unificarlos a nivel de principios del aprendizaje. El aprendizaje basado en problema si bien surge como estrategia didáctica, más claramente en escuelas de medicina, se ha aplicado en otros ámbitos educativos dando lugar a múltiples modelos. Por su parte, Graaff y Kolmos (2007) señalan, aunque hay diferencia a nivel de modelos, que existen principios de aprendizaje comunes en los modelos de ABP y que corresponden a tres aproximaciones al aprendizaje: cognitivo, contenidos y colaborativo. En la tabla 1, se describen estos principios.

Tabla 1. Principios de aprendizaje del ABP (Adaptado de Kolmos, (2009))

Aprendizaje Cognitivo	*Contenidos del Aprendizaje*	*Aprendizaje Colaborativo*
Problemas Proyectos Experiencia Contexto	Interdisciplinario Ejemplarizado Teoría y práctica Metodología de la investigación	Equipos Participación dirigida
El aprendizaje se organiza en torno a problemas y se llevará a cabo en los proyectos. El problema es el punto de partida para los procesos de aprendizaje, coloca el aprendizaje en contexto, y las bases del aprendizaje en la experiencia del alumno. Basado en proyecto significa que es una tarea que involucra análisis del problema más complejo y estrategias de resolución de problemas.	Se preocupa especialmente del aprendizaje interdisciplinario, cruza los límites entre temas relacionados y métodos. Es una práctica ejemplarizada en el sentido de que el resultado del aprendizaje es ejemplarizado para los objetivos del plan de estudios se apoya la relación entre la teoría y la práctica. El proceso de aprendizaje usa la teoría en el análisis de los problemas y de los métodos de resolución de problemas.	En el aprendizaje por equipo, el aprendizaje se lleva a cabo a través del diálogo y la comunicación. Los estudiantes no sólo aprenden el uno del otro, sino que también aprenden a compartir conocimientos y organizar por sí mismos el proceso de aprendizaje. También abarca el concepto de aprendizaje dirigido por el participante, lo que indica una propiedad colectiva del proceso de aprendizaje y, sobre todo, en la formulación del problema.

Así, se han propuesto diferentes modelos de ABP, en ellos se incluyen elementos básicos: conocimiento, aprendizaje, escenarios del problema, estudiante, rol del profesor y evaluación. Kolmos, Graaff y Du (2009), presentan los cinco modelos de ABP, basado en los estudios de Sabin-Banden (2000) y Sabin-Banden (2007), entre los que se tienen: (I) Aprendizaje basado en problemas para la competencia, (II) Aprendizaje basado en problemas para la acción profesional, (III) Aprendizaje basado en problemas para la comprensión interdisciplinaria, (IV) Aprendizaje basado en problemas para el aprendizaje trans-disciplinario y (V) Aprendizaje basado en problemas para desarrollar una actitud crítica. De los cinco modelos indicados, el que posee características

más cercanas al trabajo realizado en la formación del ingeniero informático de la UTA, es el modelo II ya que, por ejemplo, está orientado al aprendizaje en el lugar de trabajo o el escenario de los problemas a resolver y ocurren en la vida real. En la tabla 2, se muestran los elementos de dicho modelo.

Tabla 2. Elementos del modelo II del ABP (Adaptado de Kolmos et. al (2009))

Modelo II	***Descripción***
Aprendizaje basado en problemas para la acción profesional	Conocimiento: *know-how* Aprendizaje: habilidades para el lugar de trabajo. Escenario del problema: la vida real. Estudiantes: aprender a resolver problemas reales con el fin de emprender acciones prácticas Facilitadores: demostración de habilidades prácticas Evaluación: examen de aptitud para el lugar de trabajo y de los conocimientos de soporte

En la formación del ingeniero informático de la UTA, los proyectos se transformaron en el eje de la acción formativa y aparecen como una estrategia importante, pues los estudiantes: (1) deben hacer frente a información incompleta o imprecisa, (2) deben autorregularse y comprometerse con el trabajo, involucrándolos en el proceso de aprendizaje, ya que deben definir sus propios objetivos dentro de los límites impuestos, (3) deben cooperar y trabajar en grupos dividiendo la carga de trabajo entre sí e integrar las diferentes partes desarrolladas por ellos y (4) trabajar con temas multidisciplinarios (problemas complejos implican varias disciplinas). Estas habilidades prácticas son un requisito importante de las empresas y la industria, como lo señalan Macías-Guarasa et al. (2006) y Reeves et al. (2002). Por lo anterior, el ABP es relevante para el trabajo desarrollado, ya que es un método de enseñanza sistemático en que participan los estudiantes en aprendizajes de conocimientos esenciales y habilidades para la vida a través de un extendido y estructurado proceso de indagación, influenciado por el estudiante alrededor de complejas preguntas auténticas y creando productos de alta calidad. También plantean que el ABP es un enfoque de instrucción construido sobre actividades auténticas o reales. En la tabla 3, se describen las características de las actividades auténticas de aprendizajes que se logran con el ABP.

Por lo tanto, no basta con aprender cuestiones técnicas y otros temas; los conocimientos técnicos deben ser utilizados en situaciones reales. Es importante destacar las conexiones entre diferentes aspectos, tener una visión amplia de los sistemas, ilustrar las restricciones prácticas, tecnológicas y las limitaciones humanas de la resolución de problemas en el mundo real.

Así, considerando todo lo anterior y dado que uno de los requerimientos de la formación de ingenieros, en particular los informáticos “es que los estudiantes resuelvan problemas reales en contextos reales”, en un curso denominado “Ingeniería de Software”, se propuso que los proyectos sean la base del aprendizaje por parte de los estudiantes.

Un desafío importante para este curso fue enseñar sobre y en base a proyectos que los estudiantes fueran capaces de plantear proyectos de software.

Tabla 3. Características de las actividades auténticas (Adaptado de Reeves et al. (2002))

Características	***Descripción***
Relevancia del mundo real	Las actividades deben relacionarse a las actividades del mundo real de los profesionales en vez de actividades fuera de contexto o realizadas dentro de la sala de clases.
Poco definida	En las actividades se requiere que los estudiantes definan las tareas y sub tareas necesarias para cumplir con la actividad.
Tareas complejas y sostenidas en el tiempo	Las actividades se cumplen en un plazo de días, semanas o meses en lugar de minutos u horas, ya que requieren un aporte considerable de tiempo y recursos intelectuales.
Perspectivas múltiples	A los estudiantes se les brinda la oportunidad de examinar la tarea desde varias perspectivas al usar una variedad de recursos y de identificar la información pertinente de la que no lo es.
Colaboración	Para el desarrollo de la tarea, la colaboración figura como elemento integral y requerido para cumplir con ésta.
En un contexto basado en valores	Ofrece a los estudiantes la oportunidad de reflexionar y aplicar sus creencias y valores.
Interdisciplinario	Las actividades convocan perspectivas interdisciplinarias y permiten que los estudiantes desempeñen papeles diversos y logren conocimientos que se aplican más allá de una materia o de un dominio específico.
Evaluado de una forma auténtica	La evaluación se integra al aprendizaje de una manera que refleja la evaluación de calidad del mundo profesional.
Productos reales	Las actividades auténticas generan productos terminados con un valor propio en vez de un valor preparativo.
Múltiples resultados posibles	Las actividades no cuentan con una sola respuesta correcta obtenida por la aplicación de reglas y procedimientos predefinidos, sino que permiten un rango de resultados diversos, abierto a soluciones múltiples de índole original.

PLANIFICACIÓN DEL CURSO

Presentación

En este curso se presentan los principales procesos asociados con el desarrollo de proyectos de construcción de software: Iniciar, Planificar, Ejecutar, Controlar y Cerrar proyectos de Software. Así se aportará al estudiante para su formación como Ingeniero de Software, guiándolo en el desarrollo de un descripción preliminar de la problemática a abordar; en la planificación de su desarrollo; el establecimiento de las tareas propias y necesarias de la ingeniería de productos de software (especificación, análisis diseño, implementación, pruebas, y mantenimiento); en el cómo documentar y controlar sus acciones de desarrollo; y finalmente en el cómo cerrar administrativamente un proyecto.

El curso pretende aportar o tributar en la siguiente competencia específica:

- Concebir y gestionar proyectos que permitan implementar soluciones informáticas, de acuerdo a problemas específicos emergentes, en diversas áreas de negocios.

Y las siguientes subcompetencias:

- Concebir proyectos que permitan implementar soluciones informáticas.
- Planificar, estimar y controlar proyectos que permitan implementar soluciones informáticas.

Aprendizajes esperados

En la tabla 4, se describen los aprendizajes esperados para el curso de Ingeniería de Software.

Tabla 4. Descripción de los aprendizajes esperados

Resultados de aprendizaje	***Métodos***	***Evaluaciones***
Iniciar un Proyecto de Desarrollo de Software para una problemática empresarial ficticia, definiendo la necesidad de negocio que el proyecto pretende abordar, describiendo preliminarmente el producto a construir, identificando restricciones para el proyecto (por ejemplo presupuesto, plazo, personal, etc.). Planificar el Proyecto, definiendo el alcance, desarrollando un calendario de actividades, estimando costos, estableciendo un plan de garantía de calidad, definiendo una estructura organizativa y comunicativa, y definiendo los riesgos. Establecer las condiciones para la ejecución del Proyecto mediante la definición de las tareas propias y necesarias de la ingeniería de productos de software, tales como seleccionar un modelo de Proceso de Software, caracterizando las etapas de desarrollo de Software, seleccionado las herramientas para el desarrollo del software y estableciendo los mecanismos para el aseguramiento de la calidad del software a construir. Establecer las condiciones para el control del Proyecto mediante asociación del plan de garantía de calidad para el proyecto y el control de cambios en alcance, el calendario, los costos, la calidad y los riesgos; y para el control del cambio en el producto software a desarrollar definiendo un mecanismo de gestión de la configuración del software.	Resolución de ejercicios y problemas asociados con la temática Aplicación de las tareas asociadas al inicio de proyectos, generando datos e información pertinente Desarrollo de Informes escritos sobre los resultados de sus trabajos individuales Presentaciones orales de sus informes escritos utilizando recursos audiovisuales Uso de herramientas computacionales para desarrollo de algunas tareas e informes escritos y orales Uso de estándares internos e internacionales para documentar productos y subproductos Uso de normas de calidad nacional e internacional Uso de normas éticas propias de la especialidad	Pruebas Parciales Controles y ejercicios escritos Pautas de corrección controles y ejercicios Rúbricas para informes escritos, presentaciones Informes escritos sobre etapas de un proyecto de desarrollo de software

Cronograma de actividades

En la tabla 5, se describen un conjunto de actividades (se desarrollan en 16 semanas) para el curso de Ingeniería de Software.

Tabla 5. Cronograma de Actividades

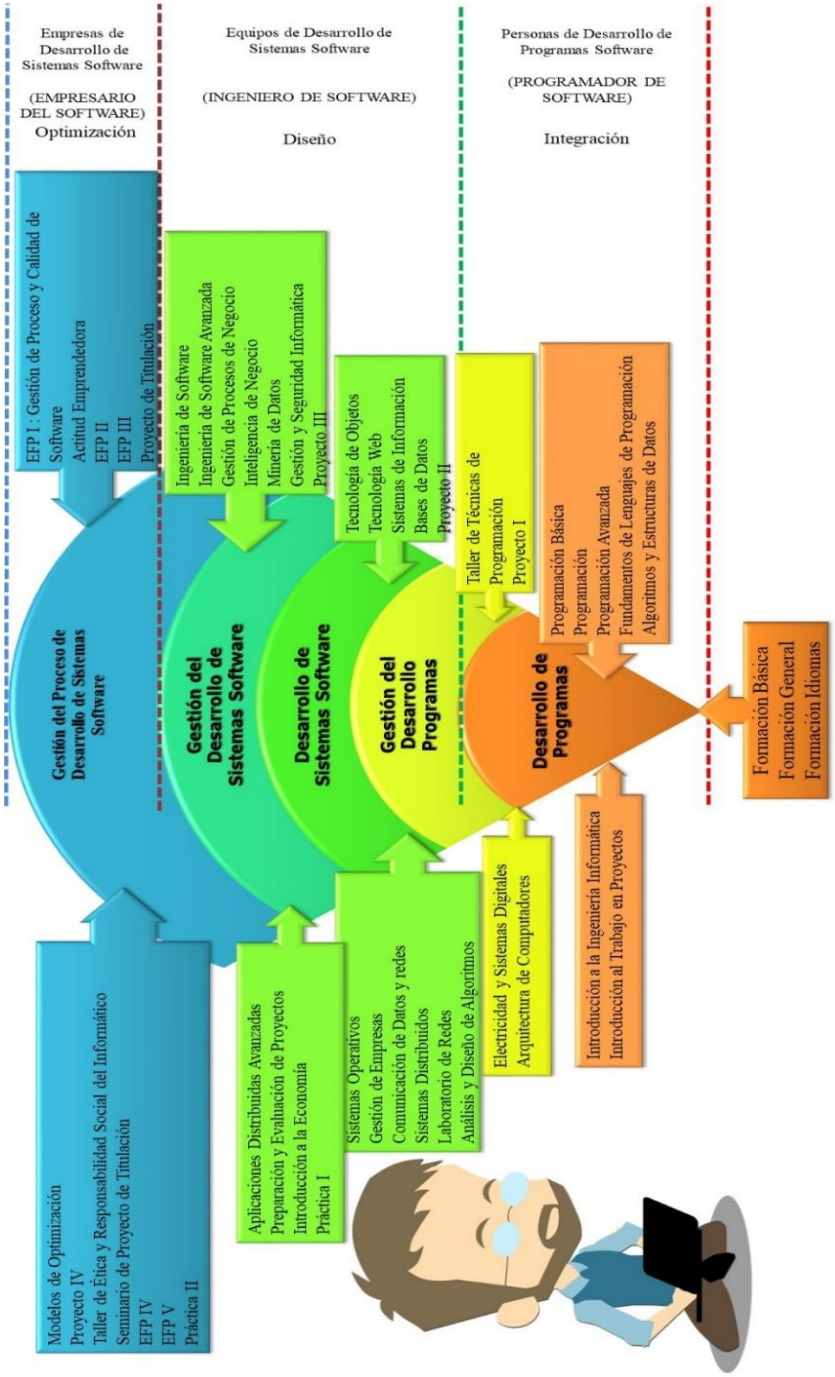
Contenidos	***Actividad de Evaluación***
Introducción • Motivación • Qué es el Software • Evolución del Software • Tipos de Software • Características o Cualidades del Software • Qué es la Ingeniería de Software • Responsabilidad Profesional • Aspectos Éticos • Proyectos de Ingeniería de Software	• Evaluación diagnóstica
Iniciar Proyectos • Necesidad de negocio que el proyecto pretende abordar • Descripción preliminar del producto • Definición del gerente de proyecto identificado/asignado • Restricciones (por ejemplo presupuesto, plazo, personal, etc.)	• Control sobre requisitos del software • Informe escrito evaluado: definición de una problemática abordable mediante un proyecto que describa la necesidad, descripción y restricciones preliminares
Planificar Proyectos • Definir el Alcance del Proyecto • Desarrollar Calendario • Planificar Organización del Proyecto • Planificar Comunicaciones • Planificar Riesgos • Estimar Costos • Calidad • Control de calidad • Plan de Garantía de Calidad	• Control sobre Calendario • Control sobre Organización • Control sobre comunicaciones • Control sobre Riesgos • Control sobre costos • Informe escrito evaluado: definición de Alcance, Calendario, Organización, Comunicaciones, Riesgos y Costos para el proyecto • Prueba Parcial • Control sobre modelos de calidad • Control sobre métricas de calidad • Control sobre técnicas de control de calidad • Informe escrito evaluado: definición de un PGC para el proyecto
Ejecutar Proyectos • Proceso de Software (Ciclos de Vida, Métodos para Desarrollar Software) • Etapas de Desarrollo de Software (Especificación de requisitos, análisis, diseño, implementación, Pruebas, Mantenimiento) • Especificación de Requisitos • Diseño: IGU • Mantenimiento • Herramientas para Desarrollar Software	• Control sobre ciclos de vida • Prueba Parcial • Control sobre especificación de requisitos • Informe escrito evaluado: definición de requisitos para el proyecto • Control sobre IGU-Metáforas • Informe escrito evaluado: definición IGU y un prototipo para el proyecto • Control sobre Mantenimiento • Informe escrito evaluado: definición PM para el proyecto • Control sobre herramientas
Controlar proyectos • Gestión de la Configuración del Software	• Control sobre GCS • Informe escrito evaluado: definición GCS para el proyecto
Cerrar Proyectos	• Control sobre Cierre • Prueba Parcial

PROCESO FORMATIVO

Una mirada personal

En la figura 1 se puede apreciar una propuesta de modelo de capas subyacente que delinea la formación como desarrollador de software del Ingeniero Civil en Computación e Informática.

Figura 1. Modelo de capas subyacente en la formación del desarrollador de software



En el modelo el estudiante recibe una formación básica, general y comunicativa indispensable para desarrollar las capacidades de abstracción e interacción personal.

Un primer nivel del modelo de capas se orienta a formar un ***programador de software***, por tanto, el currículo ofrece una orientación hacia el desarrollo de programas de software, con cursos principales como "Programación básica", "Programación", "Programación avanzada", "Fundamentos de lenguajes de programación", "Algoritmos y estructuras de datos" y cursos de soporte "Introducción a la ingeniería informática", "Introducción al trabajo en proyectos". Desde el punto de vista de formación en proyectos, este primer nivel tiene su énfasis en proyectos de integración.

Un segundo nivel con tres capas anidadas, se orienta a formar un ***ingeniero de software***, por tanto el currículo ofrece una orientación hacia el desarrollo de sistemas de software. La primera subcapa, a su vez, se orienta hacia la gestión del desarrollo de programas software. La segunda subcapa, se orienta hacia el desarrollo de sistemas software. Finalmente, la tercera subcapa se orienta hacia la gestión del desarrollo de sistemas software. Acá los proyectos se orientan hacia el diseño.

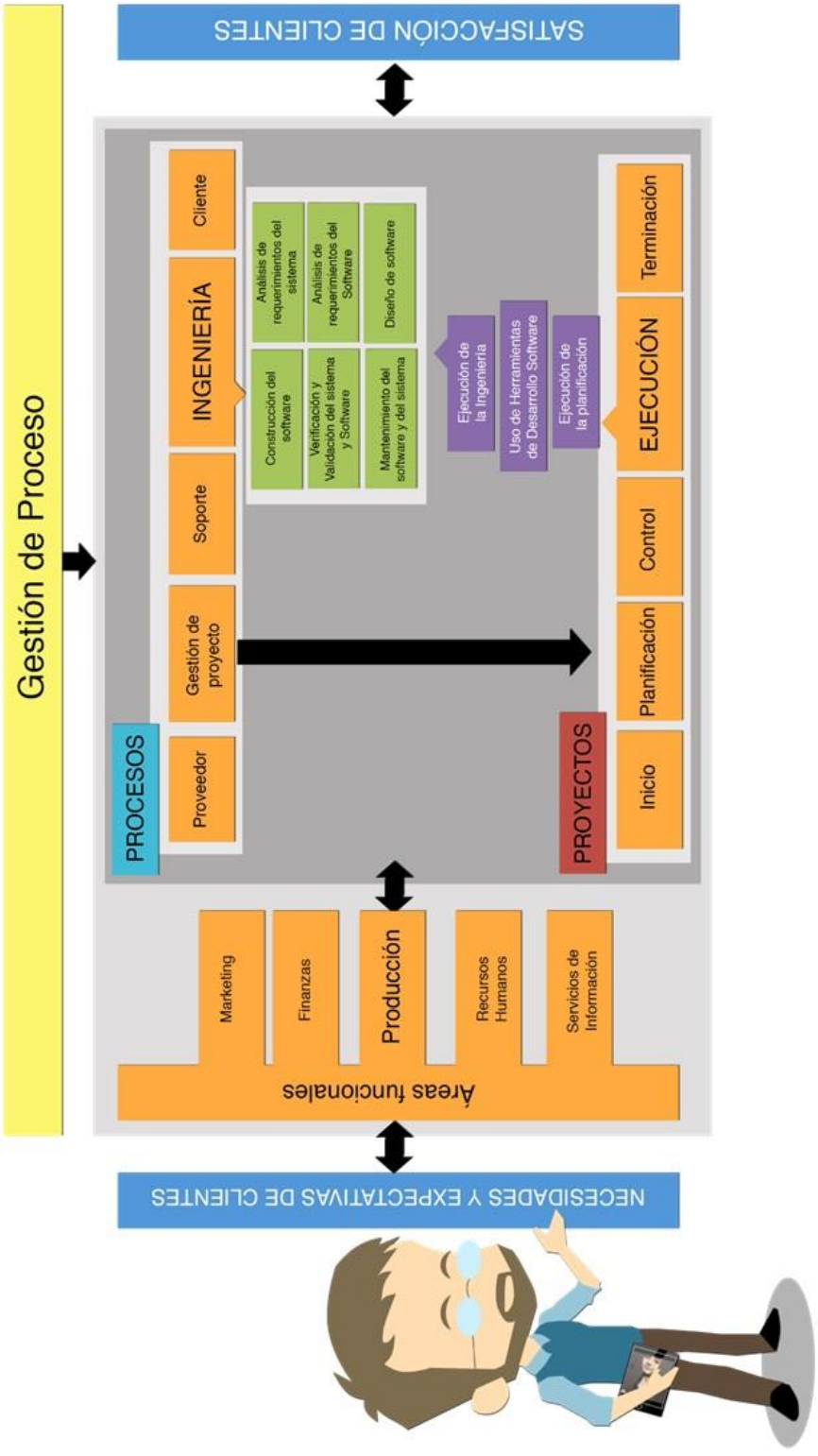
Finalmente un tercer nivel, se orienta hacia a formar un ***empresario del software***. Los proyectos se orientan hacia la optimización de los procesos para desarrollar software.

Contexto del ejercicio de la profesión

Por otra parte, otro aspecto importante a describir es la contextualización del proceso formativo con respecto al modelo de empresa. Esto porque las funciones profesionales del programador, ingeniero o empresario del software se realizan en este tipo de entidades. En la figura 2, se puede apreciar una simplificación de una empresa y la relación entre funciones, procesos y proyectos empresariales.

Una empresa en general, recibe necesidades y expectativas de clientes y mediante funciones, procesos y proyectos empresariales, logra darles satisfacción. Dentro de las funciones, se encuentra la asociada con producción que se encarga fundamentalmente, de transformar la materia prima en los servicios o productos requeridos. Esta función posee 5 áreas de proceso claves: Proveedores, Clientes, soporte, ingeniería y proyectos. Estas dos últimas son de interés del curso de Ingeniería de Software ya que se abordarán justamente los procesos asociadas con ellas: Proyectos (inicio, planificación, ejecución, control y cierre) e Ingeniería (ejecución o construcción del producto o servicio). Ambas se relacionan como se muestra en la figura 2.

Figura 2. Empresa: Funciones, Procesos y Proyectos



PIEDRAS ANGULARES

Dos conceptos o abstracciones clave son la base para el aprendizaje en Ingeniería de Software. Una de ellas son los proyectos y la otra los procesos. En la primera sección de esta primera parte, se ha explicado el por qué se ha definido el enseñar con base a proyectos a los Ingenieros Civiles en Computación e Informática.

Un proceso software (PS) es un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software. Algunas características son:

- Son complejos
- No son procesos de producción (dirigidos por excepciones; muy determinados por circunstancias impredecibles; Cada uno con sus peculiaridades)
- No son procesos de ingeniería "pura" (se desconocen las abstracciones adecuadas; dependen demasiado de demasiada gente; diseño y producción no están claramente separados; presupuestos, calendarios, calidad no pueden ser planificados de forma fiable)
- Están basados en descubrimientos que dependen de la comunicación, coordinación y cooperación dentro de marcos de trabajo predefinidos (los subproductos entregables generan nuevos requisitos; el éxito depende de la implicación del usuario y de la coordinación de muchos roles)
- Poseen 4 actividades fundamentales y comunes para todos los procesos de software: (especificación del software, desarrollo del software, validación del software, evolución del software)

De lo anterior, se puede desprender que los procesos para construir software se diferencian de otros procesos para construir otros tipos de productos. Es decir, el software se desarrolla, no se fabrica en un sentido clásico: se utiliza un modelo de proceso de desarrollo que comprende análisis, diseño, implementación y pruebas para obtener un producto de calidad.

Para comprender adecuadamente la afirmación anterior, se hace necesario describir el proceso de fabricación y compararlo con un proceso de desarrollo de software. En la tabla 6, se describen los aspectos más importantes de los procesos de fabricación.

Para comparar la fabricación de productos con el desarrollo de software, se usarán dos sectores industriales importantes para Chile. La producción de cobre y la producción de vino.

Tabla 6: Fabricación de Productos

Caracterización	***Descripción***
Definición	Un proceso de fabricación, proceso industrial, manufactura o de producción es el conjunto de operaciones (fases o etapas organizadas) necesarias para modificar las características de las materias primas (elementos de entrada, conocidos como factores), pasan a ser elementos de salida (productos), tras un proceso en el que se incrementa su valor. Las características pueden ser de naturaleza muy variada como la forma, la densidad, la resistencia, el tamaño o la estética. Se realizan en el ámbito de la industria. A veces no existe transformación física y ni siquiera se inicia desde un insumo: se trata, no de un

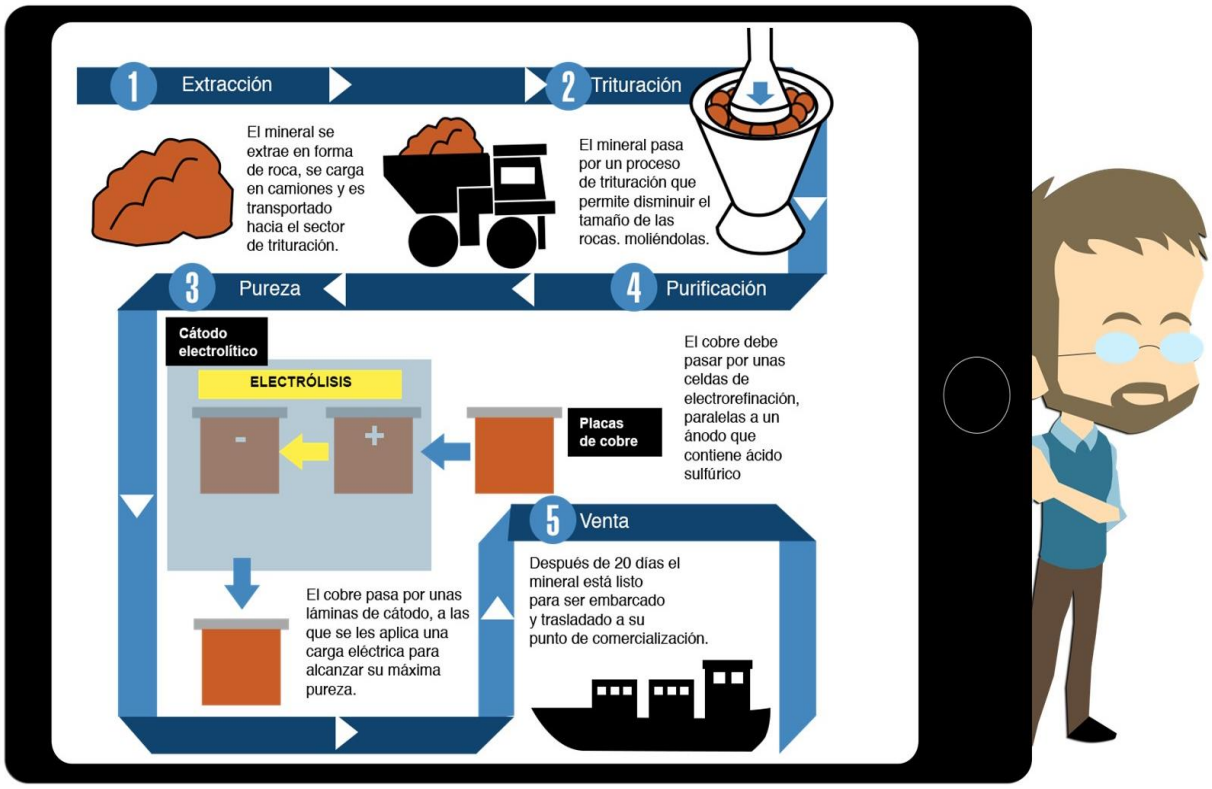
	producto, sino de un servicio.
Tipos de productos	Se puede mencionar las principales: los productos finales, que se ofertan en los mercados donde la organización interactúa, y los productos intermedios, utilizables como factores en otra u otras acciones que componen el mismo proceso de producción.
Clasificación	Pueden clasificarse de distintas formas. Según el tipo de transformación que intentan, pueden ser técnicos (modifican las propiedades intrínsecas de las cosas), de modo (modificaciones de selección, forma o modo de disposición de las cosas), de lugar (desplazamiento de las cosas en el espacio) o de tiempo (conservación en el tiempo).
Modo	Según el modo de producción, el proceso puede ser simple (cuando la producción tiene por resultado una mercancía o servicio de tipo único) o múltiple (cuando los productos son técnicamente interdependientes).
Recursos	Existen una serie de recursos que intervienen en todo proceso de producción y que deben organizarse y coordinarse adecuadamente para lograr resultados eficientes. Los recursos necesarios son los siguientes: Recursos Energéticos: electricidad, mecánica, etc. Recursos Humanos: personal capacitado. Recursos Materiales: materia prima, insumos, etc. Recursos Tecnológicos: maquinarias, herramientas, robots, etc.
Sectores	Actualmente, a la actividad productora de materia prima se la denomina SECTOR PRIMARIO, a la industria se la ubica como SECTOR SECUNDARIO, y, a la de servicios, como SECTOR TERCIARIO. Se puede destacar entonces, tres sectores productivos, a saber: Procesos Primarios: Producción de Materia Prima (industria de los insumos o explotación) Procesos Secundarios: Fabricación del Producto Tecnológico Procesos Terciario: Servicios –Comercialización del Producto Tecnológico
Sector industrial primario	Las Industrias de procesos primarios operan directamente sobre los recursos del planeta. En estas industrias de procesos primarios, aparecen tres etapas: 1. Extracción y recolección de materiales desde su fuente natural 2. Producción de la materia prima para los procesos secundarios 3. Embalaje, almacenamiento y distribución Agricultura: Cultivo de cereales, oleaginosas, forrajes, etc. Ganadería: Explotación del ganado bovino, porcino, ovino, equino, etc. Minería: Extracción de hierro, carbón, petróleo, etc. Forestal: Explotación de los bosques. Fruticultura: Producción de frutas. Piscicultura: Explotación de la pesca en ríos y mares.
Sector industrial secundario	También llamados de FABRICACIÓN. Las etapas del Proceso Productivo Secundario son: 1. Abastecimiento de insumos o Materia Prima 2. Elaboración o Fabricación del producto En esta etapa se fabrica el producto, la materia prima ingresa a la línea de producción, en la cual se le realizan una serie de operaciones de transformación hasta obtener el producto planificado. Algunas de esas operaciones pueden ser, dependiendo del producto, por ejemplo: Cortar y dar forma al producto o a sus partes, Unir, soldar, clavar o ensamblar partes, Pintar, terminar detalles, etc. Producir cambios químicos en la materia, (perfumes, esencias, medicamentos, etc.) 3. Control de calidad y evaluación de la producción 4. Envasado, Transporte y Distribución Algunas de las industrias que corresponden a este tipo de actividad pueden ser: Alimenticias: Comestibles y bebidas. Construcción: Construcción de casas, edificios, puentes, carreteras, etc. Electrónica: Fábricas de televisores, computadoras, teléfonos, etc.

Sector industrial terciario	Metalúrgica: Fábricas de automóviles, maquinarias, etc. Aeronáutica: Fábrica de aviones, helicópteros, etc.
	Este sector, también denominado sector de SERVICIOS, se dedica a la comercialización de productos ya elaborados, los que son fabricados en las industrias del sector secundario, llegan por los canales de distribución a los comercios en donde son adquiridos por los consumidores. Por lo tanto la distribución se debe efectuar de modo tal que los productos lleguen al consumidor en las cantidades solicitadas, en óptima calidad, en la forma adecuada, en el momento oportuno y en el lugar debido. Los Sistemas de Comercialización: Las empresas productoras pueden optar por realizar la distribución de los productos por cuenta propia o recurrir a los intermediarios que se encarguen de la venta de sus productos. Sistema de Comercialización DIRECTA: El fabricante llega con sus productos directamente al consumidor. Sistema de Comercialización INDIRECTA: Son Indirectos cuando el producto llega al consumidor a través de los comerciantes especializados en las actividades de venta. Estos comerciantes serían los Intermediarios entre el productor y el consumidor. Por otra parte, en este sector puede no haber un producto para comercializar, sino que hay una serie de acciones o actividades sin apariencia corpórea tales como transporte, servicios informáticos, etc.,
Formas de fabricar un mismo producto	Hay muchas formas de fabricar el mismo producto. Ahora bien, tras muchos años de experiencia, se han afianzado cuatro tipos estándar de procesos: *Job Shops*, Producción por lotes, Líneas de producción y Producción continua
Job Shops	Es un tipo de producción que permite fabricar una amplia gama de productos en series de tamaño pequeño o mediano. Los productos suelen ser conjuntos de componentes, posiblemente complicados o de alta tecnología, montados. Se utiliza para la fabricación de ciertas máquinas herramientas, robots, aviones, aeronaves y algunos prototipos. Suelen exigir mano de obra muy especializada y mucho tiempo para el diseño de los procesos y para la preparación de la maquinaria y los equipos humanos de montaje. Por todo ello, los tiempos de producción son elevados y los costos también.
Producción por lotes	Está orientada a la fabricación de lotes de tamaño medio de un determinado producto. La producción de cada lote se hace de una tirada y, una vez terminado un lote, el departamento de fabricación envía una orden de control indicando si se puede pasar a fabricar otro lote del mismo o de otro producto, en función de la demanda. La maquinaria y el personal han de estar preparados para realizar con celeridad las operaciones de cambio de lote. Es quizás el tipo de producción que se emplea para fabricar mayor número de productos. Las industrias de calzado, muebles, electrodomésticos, máquina-herramienta y otras muchas, lo utilizan.
Líneas de producción	Estos procesos son el resultado de la evolución de la producción en cadena, ideada por Henry Ford. Se utiliza para producir grandes series de unos pocos productos, que suelen estar formados mediante el montaje de piezas. El producto se desplaza colocado en cintas trasportadoras, en carros o en otros elementos de transporte y va pasando por estaciones de trabajo en cada una de las cuales se le aplica un determinado proceso. Quizás sea la fabricación de automóviles el ejemplo más típico de este tipo de producción. Se fabrican grandes series de unos pocos modelos. Otros ejemplos son la fabricación de ciertos productos de gran consumo como neumáticos, bombillas, bicicletas, envases de plástico, etc.
Producción continua	Es el tipo indicado cuando se desea producir pocos productos, de naturaleza simple (no compuestos de muchas piezas) y en grandes cantidades. Se puede ver como un flujo continuo de producto sobre el que se van realizando una serie de operaciones o procesos. Por un lado entra la materia prima y por otro sale el producto final. Este tipo de producción se aplica sobre todo en las industrias químicas, petroquímicas, textiles, de plástico y de laminación de acero.
Automatización	Automatizar un proceso es conseguir que funcione sin intervención humana. El grado de automatización deseado se suele distinguir cuatro categorías: Automatización fija, Automatización programable, Automatización flexible y Automatización total.
Automatización fija	Se utiliza cuando el volumen de producción es muy alto y, por tanto, se puede justificar económicamente el alto costo del diseño de equipo especializado para procesar el producto, con un rendimiento alto y tasas de producción elevadas. Un ejemplo típico puede ser la fabricación de automóviles. Un inconveniente de la automatización fija es que su ciclo de vida depende de la

	vigencia del producto en el mercado.
Automatización programable	Se emplea cuando el volumen de producción es relativamente bajo y hay una diversidad de productos a obtener. En este caso el equipo de producción es diseñado para adaptarse a las variaciones de configuración del producto y esta adaptación se realiza por medio de Software. Un ejemplo podría ser la fabricación de diferentes tipos de tornillos bajo pedido.
Automatización flexible	Es más adecuada para un rango de producción medio. Los sistemas flexibles poseen características de la automatización fija y de la automatización programada. Suelen estar constituidos por una serie de estaciones de trabajo interconectadas entre sí por sistemas de almacenamiento y manipulación de materiales, controlados en su conjunto por una computadora.
Automatización total	En la que, idealmente, la fabricación se realizaría sin intervención humana.

En la figura 3, se muestra los procesos para producir o fabricar cobre. Se identifican los subprocesos (fases o etapas) por las que las materias primas (mineral) son necesarias para modificar sus características y generar el producto.

Figura 3: Diagrama de proceso de fabricación del cobre



En la tabla 7, se muestra una comparación de los procesos industrial en la fabricación del cobre, vino y el proceso industrial de fabricación del software.

Tabla 7: Comparación de procesos de fabricación del vino y cobre respecto del software

Caracterización	*Cobre*	*Vino*	*Software*
Tipos de productos	Final	Final	Final / Intermedio
Clasificación	Técnico	Técnico	Técnico
Modo	Simple	Múltiple	Simple
Recursos	Energéticos, Humanos, Materiales, Tecnológicos	Energéticos, Humanos, Materiales, Tecnológicos	Energéticos, Humanos, Tecnológicos
Sectores	Primario	Secundario	Secundario (no tiene materias primas)
Formas de fabricar un mismo producto	Continua	Continua	N/A
Automatización	Parcial	Parcial	Parcial

Desde la tabla 7, se puede observar dos cosas que son relevantes. La primera es que se ha clasificado al producto software en el sector secundario, pero con la singularidad que no tiene materias primas. La segunda es que no es aplicable la característica de las formas de fabricar un mismo producto, es decir, no es posible fabricar software mediante *job shops*, producción por lotes, líneas de producción o producción continua. Por esta razón, cada vez que se desea fabricar un software se aplica un proceso de desarrollo, aunque sea un software similar, equivalente o hasta el mismo nuevamente, es decir, se reaplica el proceso una y otra vez: Análisis, Diseño, Implementación y Pruebas. Quizá se pueda re-utilizar parte o componentes existentes, pero persistirán las etapas del proceso de desarrollo.

LECTURA ADICIONAL RECOMENDADA

1. Villalobos-Abarca, Marco A., Herrera-Acuña, Raúl A., Ramírez, Ibar G., & Cruz, Ximena C. (2018). Aprendizaje Basado en Proyectos Reales Aplicado a la Formación del Ingeniero de Software. Formación universitaria, 11(3), 97-112. https://dx.doi.org/10.4067/S0718-50062018000300097.
2. Mamani, Marylin, Villalobos, Marco, & Herrera, Raúl. (2017). Sistema web de bajo costo para monitorear y controlar un invernadero agrícola. Ingeniare. Revista chilena de ingeniería, 25(4), 599-618. https://dx.doi.org/10.4067/S0718-33052017000400599.
3. Villalobos-Abarca, Marco, Karmelic-Pavlov, Vesna, & Néspolo-Cova, Mauricio. (2016). Enseñanza de los Procesos en Ingeniería Software-vs-Competitividad de Empresas Creadas por Ingenieros Informáticos. Formación universitaria, 9(1), 03-14. https://dx.doi.org/10.4067/S0718-50062016000100002.
4. Leiva Mundaca, Ignacio, & Villalobos Abarca, Marco. (2015). Método ágil híbrido para desarrollar software en dispositivos móviles. Ingeniare. Revista chilena de ingeniería, 23(3), 473-488. https://dx.doi.org/10.4067/S0718-33052015000300016.
5. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.

PARTE II:
PROYECTOS DE DESARROLLO DE SOFTWARE

CAPÍTULO 1: SOFTWARE Y PROYECTOS

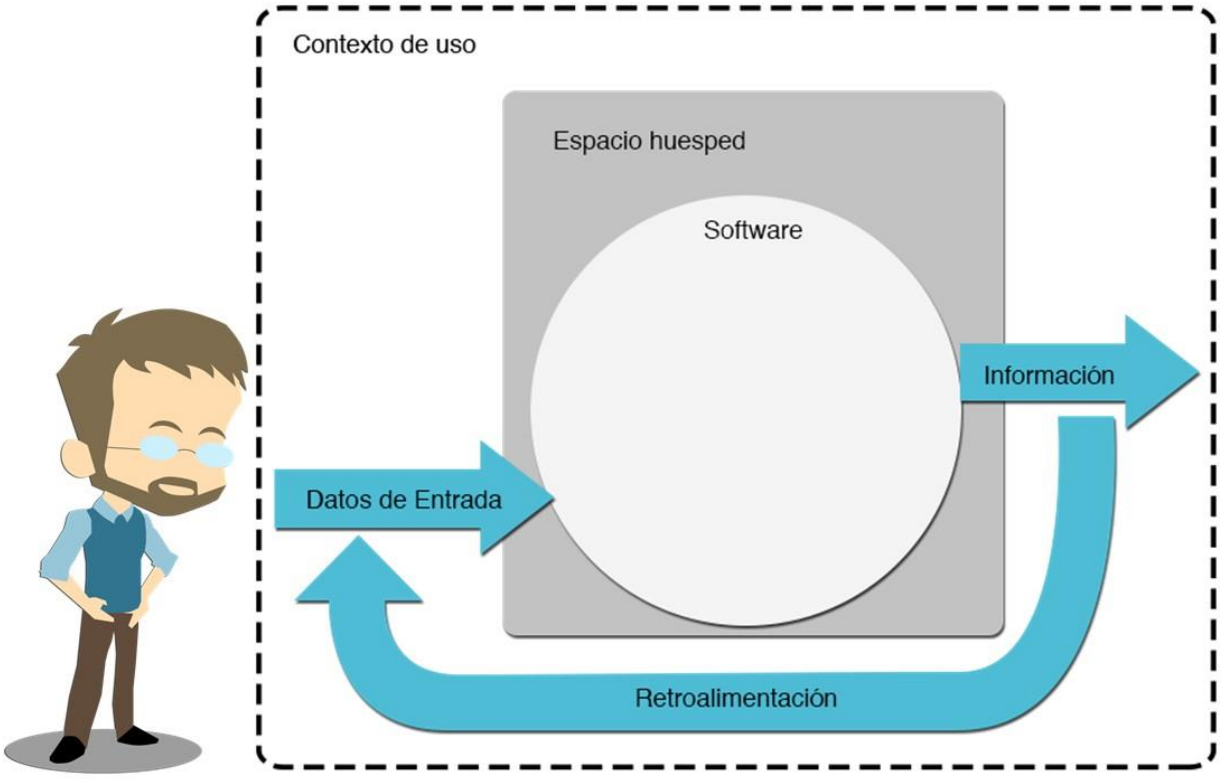
1.1 SOFTWARE

El software se ha convertido en algo fundamental para la sociedad. Es el motor que conduce a la toma de decisiones comerciales. Sirve como la base de investigación científica moderna y de resolución de problemas de ingeniería. Es el factor clave que diferencia los productos y servicios modernos. Está inmerso en sistemas de todo tipo: de transportes, médicos, de telecomunicaciones, militares, procesos industriales, entretenimiento, productos de oficina, etc.

¿Qué es el software?

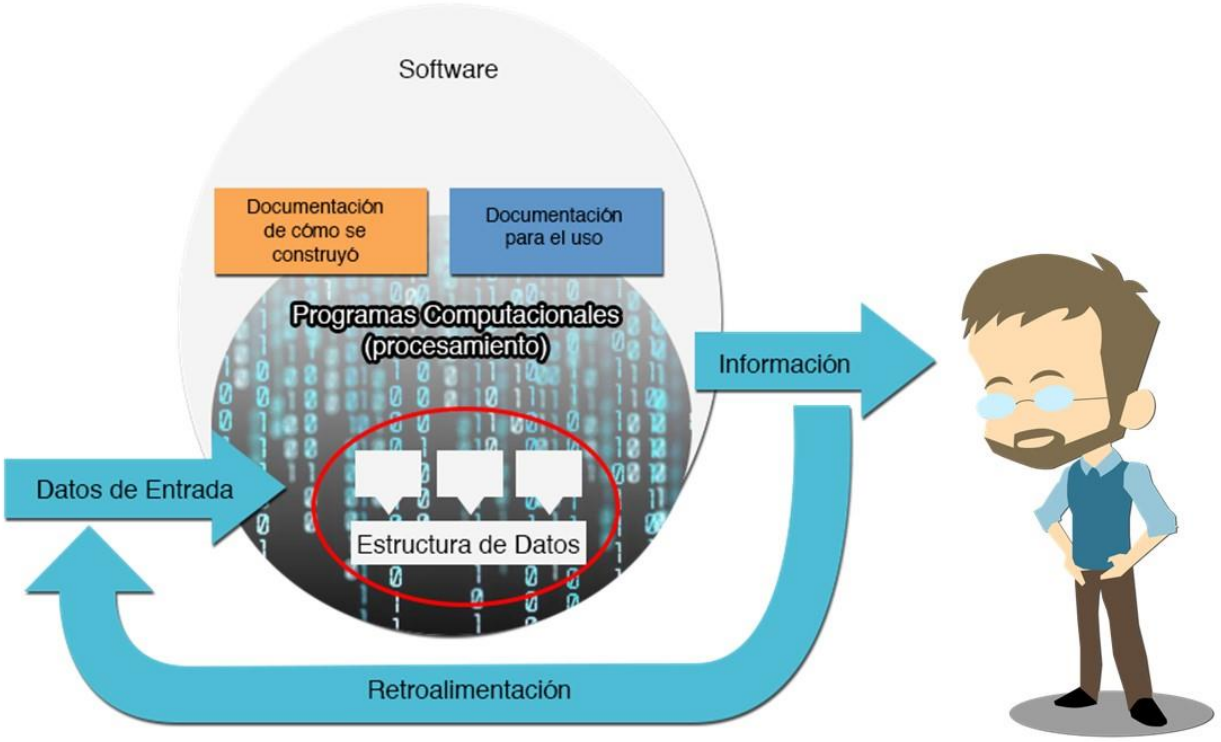
Antes de dar una definición formal, es necesario describir la utilidad de un software. En la figura 4, se muestra el uso del Software. En general el software es un producto que transforma datos de entrada en información de salida, que se encuentra alojado, por ejemplo, en un dispositivo o computadora, y que es útil en un contexto, por ejemplo, empresarial o individual.

Figura 4: Uso del Software



Ahora se puede definir un software como un producto que posee tres dimensiones: Los programas computacionales, la documentación para su uso y cómo se construyó, las estructuras de datos que le permiten manipular adecuadamente los datos de entrada para generar información de salida (Pressman, 2006; Sommerville, 2005). Esto se describe en la figura 5.

Figura 5: Qué es el Software



Tipos de software

Los productos software pueden ser de dos tipos: aquellos construidos para usuarios no determinados (genéricos) y aquellos construidos a la medida de las necesidades de usuarios determinados (personalizados). Otras taxonomías pueden ser vistas en las lecturas adicionales.

¿Cuáles son las características deseables del software?

El software posee varias características deseables que lo distinguen de otros productos, tales como el hardware. Así mismo, las características se clasifican respecto de quién lo usa (externas) o quién lo construyó (internas). En la tabla 8, se describen las características del software.

¿Qué es la ingeniería de software?

Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software (IEEE, 1993).

Ingeniería de software y ciencia de la computación

La ciencia de la computación se refiere a las teorías y los fundamentos subyacentes en los sistemas de computación. Por su parte, la Ingeniería del Software trata los problemas prácticos del desarrollo de software. Con las teorías de la ciencia de la computación no es suficiente para desarrollar software (al menos cuando el sistema tiene suficiente envergadura).

Tabla 8: Características del Software

Característica	*Descripción*
Correcto (*Correctness*)	Se comporta de acuerdo a su especificación.
Confiable (*Reliability*)	Se comporta de acuerdo con lo esperado por el usuario. A diferencia de la corrección, la confiabilidad es algo relativo. El mercado puede admitir algunos errores en el software siempre que en general se comporte en forma esperada. Sin embargo, estos errores no deben causar daños físicos o económicos.
Robusto (*Robustness*)	Se comporta en forma razonable aún en situaciones no anticipadas. Si algo se especifica como requerimiento, cumplirlo es cuestión de corrección; si no está en los requerimientos es cuestión de robustez.
Eficiente (*Eficciency*)	Si usa sus recursos, tales como memoria y el procesador (del espacio huésped) en forma económica.
Amigable (*Friendliness*)	Si sus usuarios lo encuentran fácil de utilizar.
Verificable (*Verifiability*)	Si sus propiedades pueden ser comprobadas.
Reusable (*Reusability*)	Si todo o parte de él puede ser ensamblado en uno nuevo. Debe diseñarse e implementarse para que pueda volver a usarse.
Portable (*Portability*)	Si puede ejecutarse en distintos espacios huésped o ambientes (hardware, sistemas operativos, etc.)
Interoperable (*Interoperability*)	Si puede coexistir y cooperar con otro software o sistemas.
Comprensible (*Understandability*)	Si es fácil comprender cómo funciona.
Mantenible (*Maintainability*)	Si es fácil modificarlo.

Ingeniería de software e ingeniería de sistemas

Ingeniería de Sistemas se refiere a todos los aspectos del desarrollo de sistemas que usan el computador, tanto del hardware como del software y los procesos de diseño y distribución de sistemas. La Ingeniería de Software es solo parte de este proceso.

Los ingenieros de sistemas se encargan de especificar el sistema, definir su arquitectura, integrar sus partes. Están menos relacionados con la ingeniería de los componentes del sistema (hardware y software). Al ser el software muchas veces la parte más importante del sistema, las técnicas de ingeniería del software se aplican en el proceso de ingeniería de sistemas

Relevancia de la ingeniería de software

Las economías de todos los países desarrollados dependen en gran medida del software. Cada vez más sistemas son controlados por software: comunicaciones, seguridad, administración, fábricas, comercio, agricultura, etc. el gasto en ingeniería de software, representa un alto porcentaje del producto interno bruto de los países desarrollados.

1.2 PROYECTO

Las grandes decisiones no se toman sin antes no tener un previo estudio que indique qué, cuándo, dónde, cómo y sobre todo porqué se deben llevar a cabo las acciones que se desea realizar, este análisis no es otra cosa que el inicio de un proyecto (Estrada, 2015).

¿Qué es un proyecto?

Es un emprendimiento temporario para crear un producto o servicio único. Es temporario ya que termina cuando se cumplen los objetivos; y es único porque se definen progresivamente sus características (PMBOK, 2013).

Los proyectos tienen parte interesada (*stakeholders*): involucrados (participan); interesados afectados positiva o negativamente. Algunos son: gerente del proyecto, usuarios, cliente, organización ejecutora, patrocinador, etc.

Para cada proyecto se establecen el alcance del producto (final) – sus características; y del proyecto – trabajos que se incluyen (y los que no).

Principales características de un proyecto

Todos los tipos de proyectos tienen en común una serie de características:

- Cuentan con un propósito.
- Se resumen en objetivos y metas.
- Cuentan con un alcance.
- Se han de ajustar a un plazo de tiempo limitado.
- Cuentan con, al menos, una fase de planificación, una de ejecución y una de entrega.
- Se orientan a la consecución de un resultado.
- Involucran a personas, que actúan en base a distintos roles y responsabilidades.
- Se ven afectados por la incertidumbre.
- Han de sujetarse a un seguimiento y monitorización para garantizar que el resultado es el esperado.
- Cada uno es diferente, incluso de los de similares características.

Tipos de proyecto

Existen muchos tipos de proyecto y por ello es habitual que un equipo de proyecto a menudo incluya a personas que normalmente no trabajan juntas, por proceder de organizaciones distintas o por provenir de ubicaciones geográficas diferentes. Los más comunes se indican en la tabla 9.

Gestión de Proyectos

Aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades de un proyecto, para cubrir o superar las necesidades y expectativas para un proyecto.

Procesos de un proyecto

Los procesos asociados al desarrollo de un proyecto son:

- Procesos de gestión del proyecto: refieren a describir y organizar el trabajo del proyecto.

- Procesos orientados al producto: refieren a especificar y crear el producto del proyecto definidos normalmente por el ciclo de vida del proyecto y varían por área de aplicación.
- Ambos grupos de procesos interactúan a lo largo del proyecto: por ejemplo, es imposible definir el alcance del producto sin conocimiento del área de aplicación relacionada con cómo construir el producto.

En la figura 6, se muestran los 5 procesos de gestión para un proyecto. Inicio, Planificación, Ejecución, Control y Cierre.

Tabla 9: Tipos de proyecto

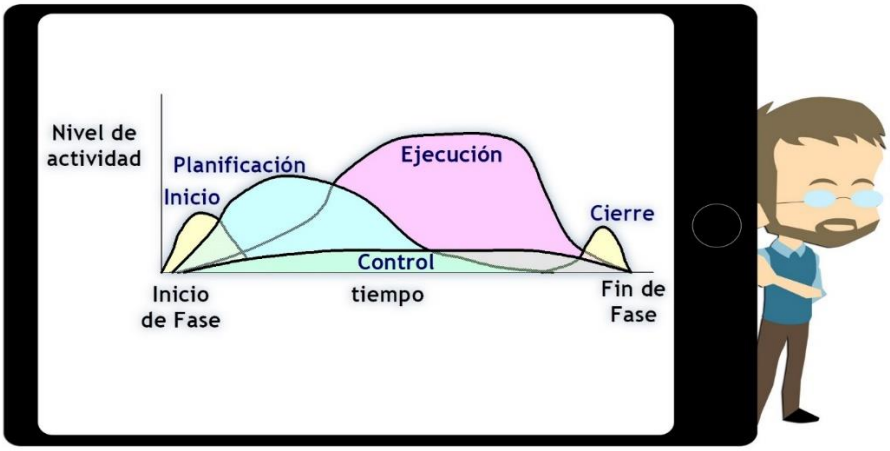
Criterio	***Tipo***
Grado de dificultad para su ejecución	Simples complejos
Procedencia del capital	Públicos Privados Mixtos
Grado experimentación del proyecto y sus objetivos	Experimentales Normalizados
Sector productivo	Construcción Energía Minería Transformación Medio ambiente Industriales Servicios
Ámbito	Ingeniería Económicos Fiscales Legales Médicos Matemáticos Artísticos Literarios Tecnológicos Informáticos
Orientación	Productivos Educativos Sociales Comunitarios Investigación
Área de influencia	Supranacionales Internacionales Nacionales Regionales Locales

Es importante señalar que existen interacciones entre los procesos. Por ejemplo, la salida de uno es entrada de otro.

Por otra parte, existe una distribución de esfuerzo por procesos, como se indica en la figura 7.

Figura 6: Procesos de gestión para un proyecto

Figura 7: Distribución de esfuerzo procesos



LECTURA ADICIONAL RECOMENDADA

1. Lea la sección 1.5 – "Mitos del Software" y la sección 6.1 – "Sistemas Basados en Computadora" del libro de Roger Pressman, 6ta edición (páginas 14-17; 134-135 respectivamente)
2. Lea la sección 1.1 – "Preguntas Frecuentes sobre la Ingeniería de Software" del libro de Ian Sommerville, 7ma edición (páginas 3-12)
3. Lea el documento sobre visión, misión y objetivos publicado en sistema de cursos UTA.
4. Lea el documento sobre la empresa VillaSoft publicado en sistema de cursos UTA.
5. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.

EJERCICIOS PROPUESTOS

1. Describa las tres dimensiones de un software. Aporte ejemplos para cada una de ellas.
2. Considerando los tipos de software vistos en clases construya una tabla donde de ejemplos de cada uno de ellos. Por ejemplo para el tipo de sistema: compiladores, editores y utilidades de gestión de archivos, ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones, etc.
3. Explique las características: externas, internas, del producto y del proceso para el software.
4. Construya una tabla para las 11 características del software, ejemplificando su significado para cada una de ellas.
5. Indique que tipo de característica son las siguientes:

Característica	Desarrollo	
	Del Producto	Del Proceso
El costo de implementación no deberá exceder de US$ 10,000.00		
El menú de navegación siempre deberá estar disponible al lado izquierdo.		
A cada usuario se le asignará un usuario del sistema y una clave, los cuales permitirán el ingreso de acuerdo un perfil determinado		
Obligar al usuario a cambiar su contraseña cada 2 meses		
Permitir que el usuario pueda cambiar la contraseña de acuerdo a las políticas de seguridad de la organización		
El sistema podrá subir la temperatura del agua por medio del regulador		
El lenguaje de programación a utilizar será Java.		
El tiempo promedio entre fallas estimado será de una vez cada seis semanas		
La aplicación se desarrollará en el Lenguaje de programación Java bajo la tecnología JAVA2EE. Con la IDE Eclipse.		
Para la gestión de base de datos se usara MySql.		
La aplicación Web deberá ser compatible con los navegadores Internet Explorer 7 y Mozilla Firefox 3.01		
El Servidor de aplicaciones será Apache Tomcat.		
El diseño de la interfaz deberá ser diseñada usando la herramienta Adobe Dreamweaver.		
No se podrán reservar libros de consulta		
La resolución mínima para una buena visualización y ejecución del sistema será un tamaño de pantalla de 1024x768 píxel.		

6. Con respecto a las cualidades del software:
   a) Una adecuada modularidad incide favorablemente en la verificabilidad.
   b) (a), y también en la eficiencia, en la facilidad de uso y de aprendizaje, y en la eficiencia en el uso.
   c) (a) y en la mantenibilidad.
   d) Todas son correctas.

7. La Ingeniería de Software se define como:
   i. La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software
   ii. Conjunto de conocimientos y técnicas cuya aplicación permite la utilización racional de los materiales y de los recursos, mediante invenciones, construcciones u otras realizaciones provechosas por el hombre
   iii. Instrucciones que cuando se ejecutan proporcionan la función y el rendimiento deseados

8. Las Ciencias de la Computación son:
   i. Las teorías y los fundamentos subyacentes en los sistemas de computación
   ii. La matemática y física del universo aplicada al desarrollo de Software
   iii. El conjunto de conocimientos específicos de la teoría de números que soportan el desarrollo del computador

9. Para el desarrollo de software es suficiente la Ingeniería de Software y las Ciencias de la Computación, ya que:

   i. Están auto-contenidos todos los conocimientos científicos y técnicos necesarios en dichas disciplinas
   ii. Otras materias como de las Ciencias Básicas, Ciencias de la Ingeniería o Ingeniería Aplicada no son redundantes como contenidos
   iii. Los profesionales de la Computación no necesitan las Ciencias Básicas, Ciencias de la Ingeniería o Ingeniería Aplicada ya que su especialidad no es una INGENIERÍA.

   [1] i
   [2] ii
   [3] iii
   [4] i y ii
   [5] ii y iii
   [6] i, ii y iii

10. Qué es un proyecto
11. Cuál es la diferencia entre Proyecto de Software, Preparación y Evaluación de Proyecto, Proyecto de Emprendimiento
12. Cuál es la diferencia entre operación y proyecto
13. Qué es la gestión de proyectos
14. Que son los "*stakeholders*". Ejemplifique
15. Qué es el alcance de un proyecto. Ejemplifique
16. Construya una tabla donde se consignen los procesos asociados a un proyecto. Indicando cuáles son subprocesos núcleo y facilitadores.
17. Explique la figura 7 del texto.

TRABAJO DE PROYECTO

Aplicación práctica 1

La empresa "Villa Soft" de desarrollo de software (memoria 2015) ha asumido el desarrollo de varias iniciativas para el presente año.

Se la ha pedido como integrante de "Villa Soft" se constituya algún equipo que definirá un **proyecto** para cada producto y/o servicio. Así, como integrante de algunos de los equipos que elaborará cada proyecto, deberá hacer que ocurran las siguientes etapas descritas en la siguiente tabla.

ETAPA	DESCRIPCIÓN
Iniciar	Reconocer que el proyecto (o fase) debe comenzar y comprometerse a ello.
Planificar	Concebir y mantener un esquema factible de trabajo para satisfacer la necesidad de negocio por la que se encara el proyecto.
Ejecutar	Coordinar las personas y otros recursos para ejecutar el plan.
Controlar	Asegurar que los objetivos del proyecto mediante control y medición del progreso y si es necesario, tomando acciones correctivas.
Cerrar	Formalizar la aceptación del proyecto (o fase) y llevarla a un fin ordenado.

Constituido cada equipo, se nombrará a un Jefe de Proyecto y se elaborará un primer informe de trabajo que dé cuenta de las acciones y decisiones tomadas para la primera etapa (**inicio**) del desarrollo del respectivo proyecto.

Por lo anterior, se requiere la elaboración de un informe según el siguiente formato (sólo la sección de desarrollo, ya que el resto está determinado). El Jefe de Proyecto que reportará directamente al Gerente de Jefes de Proyecto.

III. DESARROLLO

3.1 Descripción de la Empresa "Villa Soft"
- 3.1.1 Visión
- 3.1.2 Misión
- 3.1.3 Historia
- 3.1.4 Productos
- 3.1.5 Factor humano
- 3.1.6 Organigrama
- 3.1.7 Proveedores
- 3.1.8 Ventas
- 3.1.9 Servicios de posventa
- 3.1.10 Infraestructura y tecnología de información
- 3.1.11 Unidades de producción

3.2 Descripción de la Empresa dueña del Proyecto “NN”

- 3.2.1 Visión
- 3.2.2 Misión
- 3.2.3 Historia
- 3.2.4 Productos
- 3.2.5 Factor humano
- 3.2.6 Organigrama
- 3.2.7 Proveedores
- 3.2.8 Ventas
- 3.2.9 Servicios de posventa
- 3.2.10 Infraestructura y tecnología de información
- 3.2.11 Unidades de producción

CAPÍTULO 2: INICIAR UN PROYECTO

2.1 PROCESO DE INICIO

El objetivo fundamental es comprometer a la organización para comenzar la siguiente fase del proyecto.

Las salidas de este proceso son:

- Definición del proyecto (*project charter*): documento que reconoce formalmente la existencia del proyecto; debiera incluir (directamente o por referencia):
    - ✓ la necesidad de negocio que el proyecto pretende abordar
    - ✓ la descripción del producto
    - ✓ podría ser el contrato en caso de que lo haya
- Gerente de proyecto identificado/asignado
- Restricciones (por ejemplo presupuesto, plazo, personal, etc.)

2.2 SUPUESTOS Y RESTRICCIONES EN PROYECTOS

En los proyectos se debe tomar en cuenta que no tienen total certeza de los eventos que pueden ocurrir durante su ciclo de vida. Es por ese motivo que se deben estimar los diferentes supuestos y restricciones para determinar el camino a seguir de un proyecto.

Según la OCIO (*Office of the Chief Information Officer Washington State*) los supuestos son circunstancias y eventos que deben ocurrir para que el proyecto sea exitoso, pero que no están dentro del control del equipo del proyecto. Los supuestos son siempre aceptados como verdaderos a pesar de no ser demostrados.

Según la OCIO, las restricciones son aquellos elementos que restringen, limitan o regulan el proyecto y, al igual que los supuestos, no están en control del equipo del proyecto.

Existen dos tipos de restricciones:

- Del negocio: Generalmente se enfocan en el tiempo, dinero y recursos disponibles para el proyecto.
- Técnicas: Generalmente se enfocan en decisiones de arquitectura (esto aplicado a proyectos de Tecnologías de información).

En la tabla 10 se muestra las similitudes y diferencias entre estos dos términos:

Tabla 10: Similitudes y diferencias entre las restricciones del negocio y las técnicas

	Supuestos	***Restricciones***
Característica	Condición, Circunstancia o evento.	Condición, Circunstancia o evento.
Impacto	Permite al proyecto proceder.	Restringe y limita la ejecución del proyecto.
Proceso	Debe ser analizado y monitoreado para asegurar validez y relevancia a medida que el proyecto procede.	Debe ser identificado e incorporado en los planes del proyecto para asegurar que este sea realista.

Desde el inicio hasta el final, los supuestos y restricciones establecen el escenario para la planificación y ejecución del proyecto. A medida que el proyecto es planificado, los supuestos y restricciones se usan para definir tareas, cronogramas, asignación de recursos y distribuir presupuesto.

2.3 FUNCIONES DEL GERENTE DE PROYECTO

El trabajo de un gerente de proyectos se relaciona con las actividades administrativas de planificación, organización, dirección y control de los recursos a su cargo (personal, presupuesto, equipo y materiales) para satisfacer los requerimientos técnicos, de costo y de tiempo, que permitan finalizar con éxito el o los proyectos bajo su responsabilidad, según se haya presupuestado.

Roles:

Para cumplir con las anteriores responsabilidades, el gerente de proyecto debe asumir los siguientes roles:

- Planificador: tanto del proyecto como de los recursos a su cargo.
- Integrador: de los esfuerzos de las distintas áreas de una empresa que participan en el proyecto.
- Comunicador: para mantener el interés por el proyecto y la oportuna acción de las diferentes áreas de la empresa.
- Administrador: de recursos físicos, tecnológicos, humanos y financieros pertenecientes al proyecto.
- Mentor (coach): para capacitar, estimular, supervisar, motivar y corregir a los integrantes de su equipo del proyecto.

Funciones:

- Definir los objetivos del proyecto: que sean claros y alcanzables según las capacidades de la empresa.
- Alinear el proyecto con la estrategia empresarial / institucional.
- Manejar los recursos físicos, financieros, humanos y su asignación a las tareas.
- Administrar los costos y presupuestos.
- Administrar la calidad del proyecto según los estándares de desempeño definidos.
- Vigilar que las tres restricciones (calidad, costo y tiempo) a que se enfrentan todos los proyectos se gestionen adecuadamente.
- Gestionar los plazos para lograr terminar el proyecto a tiempo.

- Participar en la integración del equipo del proyecto: definir los perfiles con las competencias requeridas.
- Garantizar que el personal del proyecto reciba toda la formación necesaria.
- Analizar y manejar los riesgos.
- Administrar el recurso humano.
- Manejar las comunicaciones.
- Informar a todos los actores del proyecto sobre los avances o retrasos.
- Orientar y/o delegar a su equipo, ejerciendo la supervisión necesaria.
- Negociar con proveedores externos para asegurarse de que todos los materiales necesarios para un proyecto estén en el momento adecuado.
- Manejar las herramientas, los métodos, las métricas y los cronogramas del proyecto.
- Hacer seguimiento y control oportuno.
- Administrar los problemas y los cambios que el proyecto exija sobre la marcha.

LECTURA ADICIONAL RECOMENDADA

1. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.
2. David Fuller Padilla, Apuntes de Taller de Ingeniería de Software, Capítulo 4: Roles en el desarrollo de software, 2003. Publicado en sistema de cursos UTA.
3. Carlos Alberto Mejía Cañas, Publicación periódica coleccionable, Las funciones de un gerente de proyectos, Nº 1108, DOCUMENTO PLANNING. Publicado en sistema de cursos UTA

EJERCICIOS PROPUESTOS

1. Indique cuál es la salida de la tarea "Definición del proyecto"
2. Para que puede ser útil la "Definición del Proyecto"
3. Qué es una restricción y una limitación. Cuál es la diferencia. Ejemplifique
4. Indique cuáles podrías ser restricciones típicas en un proyecto de desarrollo de software

TRABAJO DE PROYECTO

Aplicación práctica 2

Se le solicita escribir el Chárter para su proyecto en los siguientes términos.

III. DESARROLLO

3.1 Definición chárter del proyecto

3.1.1 El problema o necesidad
3.1.2 Porqué es importante abordar el proyecto
3.1.3 Descripción (alcance) preliminar del producto (software) y proceso
3.1.4 Restricciones (presupuesto, plazo, personal, tecnologías, métodos, etc.)

Donde:

Problema o Necesidad:

- Se debe describir el problema en términos concretos, explícitos y específicos.
- Un problema correctamente planteado está parcialmente resuelto, a mayor exactitud corresponden más posibilidades de obtener una solución satisfactoria.
- El desarrollador debe ser capaz, no sólo de conceptualizar el problema, sino también de verbalizarlo en forma clara, precisa y accesible.
- En algunas ocasiones se sabe lo que se desea hacer, pero no se sabe o puede comunicarlo a los demás, y es necesario que se realice un esfuerzo por traducir su pensamiento a términos que se entienda y acepte para después poder comunicarlo a los demás.
- El planteamiento del problema es la fase más importante en todo el proceso de desarrollo, de ahí que muchos autores hagan referencia que un problema bien planteado es la mitad de su solución.

- ✓ El planteamiento del problema debe dejar bien establecido: Identificar claramente la(s) pregunta(s) que se quiere(n) resolver y el/los problema(s) concreto(s) a cuya solución o entendimiento se contribuirá con la ejecución del proyecto de desarrollo.

Porqué es importante abordar el proyecto

- Ofrece una descripción precisa y completa de la naturaleza y magnitud del problema y justificar la necesidad del desarrollo, en términos del crecimiento del país, región, comuna o empresa.
- Este ítem debe responder claramente a la pregunta: ¿Por qué se debe realizar este proyecto?. Tener en cuenta que no se debe exponer la solución prevista al problema.

Descripción (alcance) preliminar del producto (software) y proceso

Características preliminares del producto

- Funcionalidades que ofrecerá el producto (Gestión de Proveedores, Clientes, etc.)
- Funciones de seguridad
- Funciones interfaz de usuario
- Etc.

Características preliminares del proceso

- Tipo de proceso
- Duración del proceso
- Etapas del proceso
- Etc.

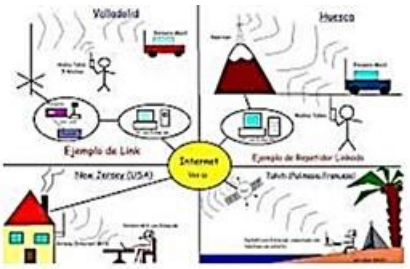
Esquema preliminar de la solución propuesta

Una forma global de presentar una solución es un esquema general, que relaciona todos los componentes, módulos, subsistemas, conceptos, ideas, etc. que son parte de la solución. Ejemplos de diagramas se muestran en la siguiente figura.

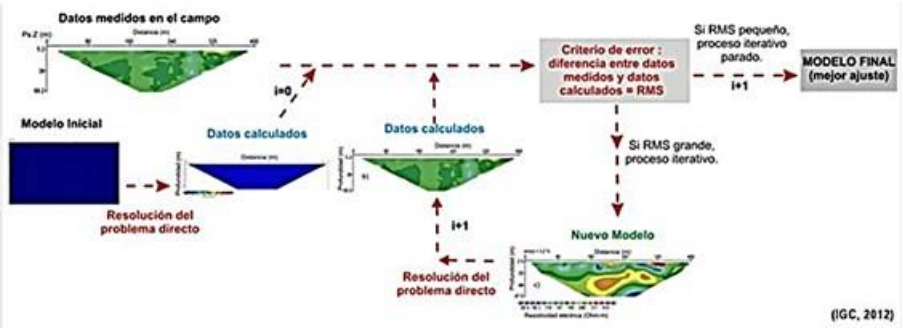
Restricciones

- Presupuesto
- Plazo
- Personal
- Tecnologías
- Métodos


Monitoreo Satelital Unidad o Mercancía
Custodia Virtual
Monitoreo Satelital de patrulla
Monitoreo de Video Vigilancia
Solución Integral e Inteligente
Patrulla y Custodio para unidad y/o Mercancía en tránsito
Testigo ocular físico


Valladolid
Huesca
Ejemplo de Link
Internet
Ejemplo de Repetidor Linkado


PROBLEM
?
IDEA
!
SOLUTION

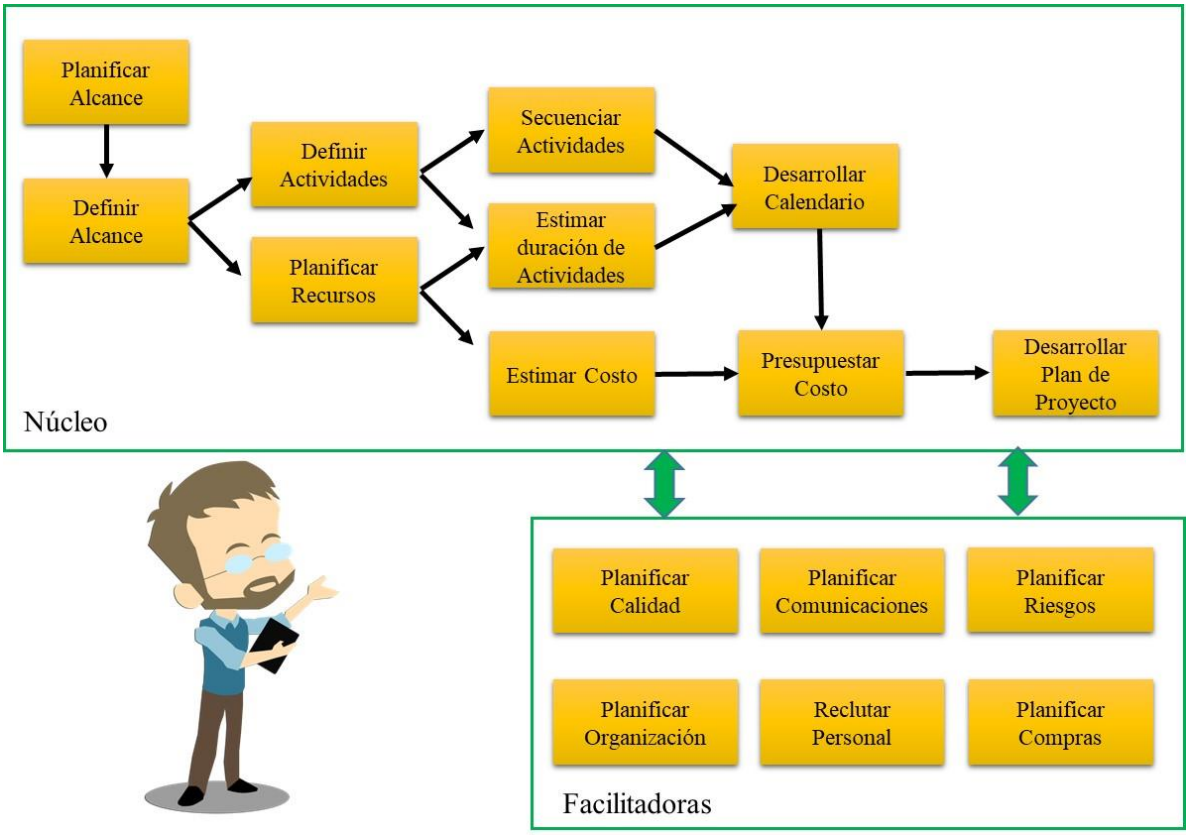

Datos medidos en el campo
Criterio de error : diferencia entre datos medidos y datos calculados = RMS
Si RMS pequeño, proceso iterativo parado.
MODELO FINAL (mejor ajuste)
i+1
i=0
Modelo Inicial
Datos calculados
Datos calculados
Si RMS grande, proceso iterativo.
Resolución del problema directo
i+1
Nuevo Modelo
Resolución del problema directo
(IGC, 2012)

CAPÍTULO 3: PLANIFICAR UN PROYECTO

3.1 PROCESO DE PLANIFICACIÓN

En la figura 8, se muestra las tareas asociadas con el proceso de planificación de un proyecto.

Figura 8: Tareas del proceso de planificación



Tareas Núcleo

- Planificación de alcance: definir (por escrito) el alcance, base para decisiones futuras en el proyecto.
- Definición del alcance: subdividir los principales resultados (“entregables”) en componentes menores.
- Definición de actividades: identificar las actividades específicas necesarias para producir los resultados del proyecto.
- Secuencia de actividades: identificar y documentar las dependencias entre actividades.
- Estimar duraciones: estimar número de períodos de trabajo necesarios para cada actividad.
- Desarrollar calendario: analizar la secuencia de actividades, duraciones y requerimientos de recursos, para generar el calendario.
- Planificación de recursos: determinar qué recursos (personas, equipos, materiales) y las cantidades se van a precisar para llevar a cabo las actividades.

- Estimar costo: desarrollar una aproximación a los costos de los recursos necesarios.
- Presupuestar costo: asignar la estimación global a cada ítem de trabajo.
- Desarrollar el plan del proyecto: tomando los resultados de otros procesos de planificación para dejarlos de forma consistente y coherente en un documento.

Tareas facilitadoras

- Planificar calidad: identificar los estándares de calidad relevantes para el proyecto y determinar cómo satisfacerlos (métricas, PGC).
- Planificación de la organización: Identificar, documentar y asignar roles de proyecto, responsabilidades y relaciones de dependencia y comunicación.
- Reclutar personal: Obtener recursos humanos necesarios, asignarlos e incorporarlos al trabajo.
- Planificar comunicaciones: determinar las necesidades de información y comunicaciones para las partes interesadas: quién precisa qué información, cuándo y cómo se le hará llegar consistente y coherente en un documento.
- Planificar Riesgos: Identificación de Riesgos (determinar y documentar los riesgos que probablemente afecten el proyecto); Cuantificación de Riesgos (evaluar los riesgos y sus interacciones para evaluar la severidad del impacto; Desarrollar respuesta a los riesgos (definir mejoras para las oportunidades y respuestas a los riesgos).
- Planificar Compras: Determinar qué y cuándo comprar.

3.2 CALENDARIO DE UN PROYECTO

Diagrama de Gantt

El diagrama de Gantt es una herramienta para planificar y programar tareas a lo largo de un período determinado. Gracias a una fácil y cómoda visualización de las acciones previstas, permite realizar el seguimiento y control del progreso de cada una de las etapas de un proyecto y, además, reproduce gráficamente las tareas, su duración y secuencia, además del calendario general del proyecto.

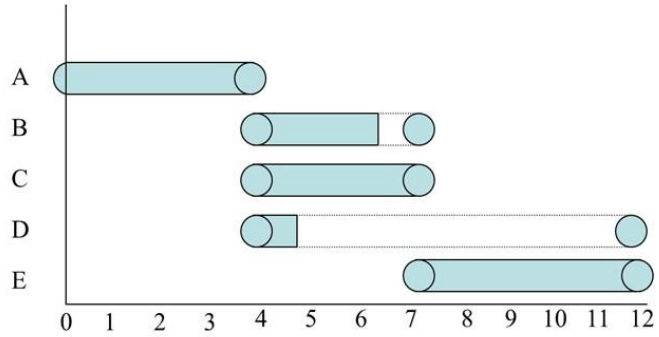
Algunas consideraciones para elaborar un Diagrama de Gantt son:

a. Hacer una lista de todas las actividades que puede requerir un proyecto. Puede que, como resultado, se obtenga una lista demasiado larga. Sin embargo, a partir de esto se definirá tiempos para la realización de cada tarea, prioridades y orden de consecución. Además, se agruparán las actividades por partidas específicas para simplificar al máximo la gráfica.
b. El diseño del diagrama de Gantt debe ser lo más esquemático posible. Debe transmitir lo más importante, ya que será consultado con frecuencia. Las personas implicadas en el proceso deben quedarse con una idea clara de lo que está sucediendo en un momento concreto del proceso.
c. Si se desea, se puede crear y mantener actualizada otra versión más detallada para la persona que ejecuta el proyecto. Gracias al diagrama de Gantt, es posible una monitorización clara del progreso para descubrir con facilidad los puntos críticos, los períodos de inactividad y para calcular los retrasos en la ejecución. De este modo, ayuda a prever posibles costos sobrevenidos y permite reprogramar las tareas de acuerdo a las nuevas condiciones.
d. Finalmente, cabe decir que por su sencillez, facilidad de uso y bajo costo se emplea con mucha frecuencia en pequeñas y medianas empresas.

Ejemplo: Construcción de una casa

Figura 9: Ejemplo diagrama Gantt

Activ	Descripción	Predecesor	Durac. (sem)
A	Cimientos, paredes	-	4
B	Plomería, electricidad	A	2
C	Techos	A	3
D	Pintura exterior	A	1
E	Pintura interior	B, C	5



Ruta crítica

La Ruta Crítica es la ruta más larga a través de una red que representa un conjunto de actividades de un proyecto. Determina la longitud del proyecto. Toda red tiene al menos una ruta crítica. Es posible que haya proyectos con más de una ruta crítica.

El procedimiento para determinar la ruta crítica puede ser visto en el anexo procedimientos del presente documento.

3.3 COSTOS Y BENEFICIOS DE PROYECTOS INFORMÁTICOS

Clasificación

Una clasificación básica es mediante la tangibilidad e impacto. Los costos o beneficios tangibles y de impacto directo, son fáciles de valorar y por lo tanto pueden ser explícitamente cuantificados o traducidos a términos monetarios. Por el contrario los intangibles y de impacto directo, son generalmente difíciles de cuantificar, por lo que comúnmente son representados en términos cualitativos. Por ejemplo, mejorar el ambiente de trabajo podría tener un impacto directo sobre el área de aplicación del proyecto, sin embargo su tangibilidad puede ser compleja. Así, la representación de costos o beneficios en este caso, normalmente se basa en interpretaciones subjetivas.

Con respecto a los costos o beneficios tangibles o intangibles, pero de impacto indirecto suelen ser los más difíciles de valorar en términos monetarios, en cuyo caso se usan parámetros o interpretaciones subjetivas. Un caso típico tiene que ver con los costos asociados a proyectos que pudieran generar problemas ambientales.

Los costos pueden ser fijos y variables. Normalmente exceden lo planificado. La vida útil de los proyectos es volátil. Nuevas tecnologías, nuevos, cambios de requerimientos, etc. hacen que la vida útil sea compleja de determinar. Dado el alto nivel de competencia, se subestiman los costos.

En las tablas 11 y 12 se muestran algunos costos y beneficios asociados con proyectos informáticos.

Tabla 11: Costos

Tangibles y directos	*Intangibles y directos*	*Tangibles e indirectos*	*Intangibles e indirectos*
Inversión inicial ✓ Estudios ✓ Hardware y software ✓ Comunicaciones ✓ Instalaciones ✓ Capacitación ✓ Etc. Costos de operación ✓ Sueldos ✓ Mantención ✓ Rentas ✓ Energía ✓ Insumos ✓ Etc.	• Servicios adicionales • Relaciones humanas • Administración • Etc.	• Retrasos en la puesta en marcha • Demoras por aprendizajes • Etc.	• Resistencia al cambio • Problemas organizacionales • Costos medioambientales • Etc.

Tabla 12: Beneficios

Tangibles y directos	*Intangibles y directos*	*Tangibles e indirectos*	*Intangibles e indirectos*
• Mejoramiento en la productividad de los trabajadores • Ahorro de horas hombre o trabajadores • Etc.	• Mejoramiento de la gestión • Mejoramiento en la productividad organizacional • Etc.	• Mejoramiento de la productividad del capital • Ahorro en costos de operación • Ahorros por arriendos o venta de información • Etc.	• Calidad de la información • Imagen de la compañía • Etc.

3.4 COSTOS DE DESARROLLO DE UN PRODUCTO SOFTWARE

Estimación de costos

La estimación de costos es una aproximación a los costos de desarrollo del software. Se mide en términos de esfuerzos requeridos por persona/mes por año. El proceso de estimación de costos es un conjunto de técnicas utilizadas para desarrollar estimaciones de costo del software a partir de entradas, limitaciones y otros factores que puedan influir.

El costo de desarrollo del software más importante es el pago a los ingenieros de software (costo de esfuerzo).

Técnicas de estimación

- Modelado de algoritmo de costo: Modelo en función de información histórica relacionada con métricas de software y su costo. Así se predice el esfuerzo requerido.
- Opinión de expertos: Cada uno opina, se compara y se intenta llegar a una conclusión.
- Estimación por analogía: Aplicable cuando se desarrollaron proyectos similares.
- Asignar precios para ganar. El costo del software se estima independientemente de lo que el cliente esté dispuesto a pagar por él. El esfuerzo depende de lo que el cliente quiera pagar y no de la funcionalidad del software.

Modelado algorítmico de costos

Enfoque más sistemático aunque no el más preciso. Se construye analizando los costos y atributos de proyectos realizados. Muchas componentes de estimación algorítmica y exponencial. El costo del proyecto no se incrementa linealmente con su tamaño.

Generalmente, los modelos de estimación del esfuerzo consisten en dos partes:

- P1: Una base, para el cálculo, como función del tamaño del software y es de la siguiente forma: $E = A + B \times (KLDC)^c$. Donde E es la estimación del esfuerzo en hombre – mes (H – M). A, B, y C son constantes; y KLDC es el número estimado de miles de línea de código en el sistema final.
- P2: Un proceso de ajuste (modificar la base de estimación para calcular la influencia de los factores ambientales). Factores típicos son:

  - ✓ Atributos del producto (Fiabilidad, complejidad, etc.)
  - ✓ Atributos de hardware (Velocidad de ejecución, restricciones de memoria, tiempo de respuesta, etc.)
  - ✓ Atributos del personal (Capacidad de análisis, capacidad con el lenguaje, planificación temporal, etc.)
  - ✓ Atributos del proyecto (Utilización de herramientas de software, métodos, etc.)

Técnica de estimación de costos por puntos de casos de uso

Desarrollado por Gustav Karner en el año 1993. Basado en el método de Puntos de función, cuyo objetivo principal es estimar cuantas horas son necesarias para desarrollar el software y a partir de eso, determinar la cantidad de personas requeridas.

El método depende de casos de uso bien estructurados y bien escritos, con un nivel conveniente de detalle textual. En general, desarrollos con largos casos de uso complicados, emplean mayor esfuerzo para diseñar e implementar que desarrollos, cuyos casos de uso son menos complicados. Además el tiempo para completar un desarrollo está afectado por lo siguiente:

- El número de pasos para completar el caso de uso.
- El número y complejidad de los actores.
- Los requerimientos técnicos de los casos de uso, como la concurrencia, seguridad y la

eficiencia.
- Varios factores del entorno tales como la experiencia y conocimiento del equipo de desarrollo.

Así, el método analiza los actores de casos de uso, los escenarios, y varios factores técnicos y de ambiente y abstrae todo ello en una ecuación. La ecuación de los UCP está compuesta por tres variables:

- Los Puntos de casos de uso sin ajustar (UUCP).
- El factor de complejidad técnica (TCF).
- El factor de complejidad del entorno (ECF).

Cada variable es definida y calculada de forma separada usando pesos, valores subjetivos y constantes restrictivas. Adicionalmente cuando la productividad es incluida como un coeficiente que se expresa en tiempo, la ecuación puede ser usada para estimar el número de horas hombre para completar el proyecto.

El procedimiento para calcular el costo mediante el método de Karner, puede ser visto en el anexo procedimientos del presente documento.

3.5 ESTRUCTURA ORGANIZATIVA DE UN PROYECTO

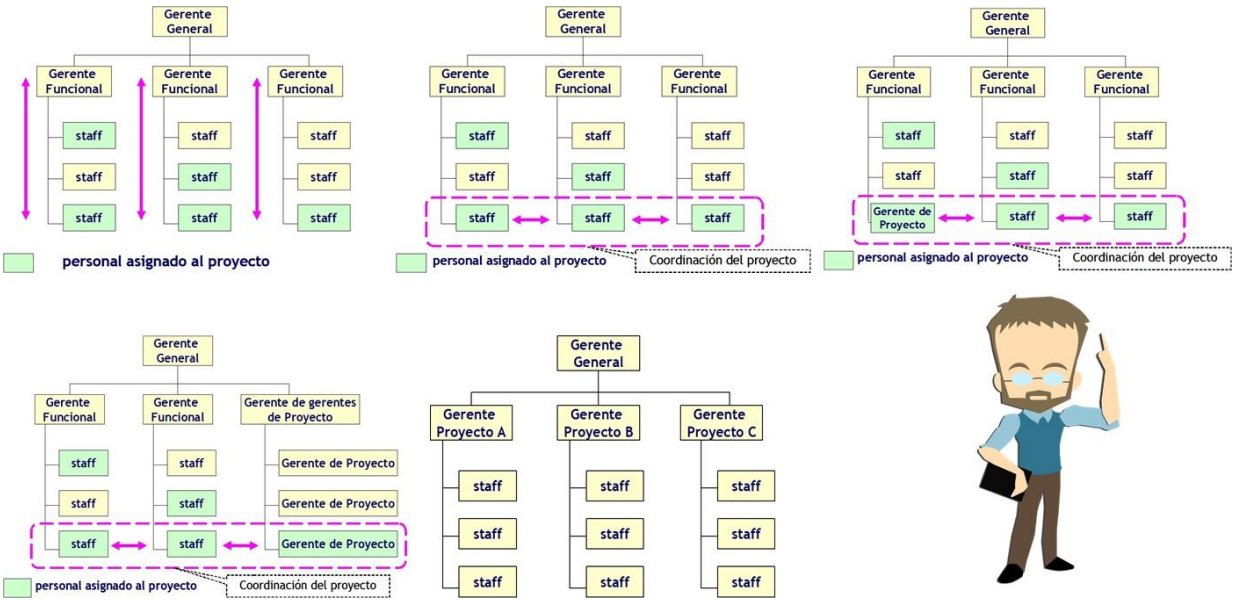
Los proyectos pueden ser organizados bajo las estructuras que se indican en la tabla 19, con sus respectivas características v/s tipo de organización.

Tabla 19: Características v/s tipo de organización (PMBOOK, 2013)

Características proyectos	***Tipo organización***				
	Funcional	***Matricial débil***	***Matricial balanceada***	***Matricial Fuerte***	***Proyectizada***
Autoridad del gerente de proyecto	Poca o ninguna	Limitada	Baja a moderada	Moderada a alta	Alta a casi total
% de personal asignado al proyecto tiempo completo	Casi ninguno	0-25%	15-60%	50-95%	85-100%
Dedicación del gerente de proyecto	*Part-time*	*Part-time*	*Full-time*	*Full-time*	*Full-time*
Títulos comunes para el gerente de proyecto	Coordinador/Líder del proyecto	Coordinador/ Líder del proyecto	Gerente/ Jefe de Proyecto	Gerente de Proyecto/ Programa	Gerente de Proyecto/ Programa
Personal administrativo para la gestión del proyecto	*Part-time*	*Part-time*	*Part-time*	*Full-time*	*Full-time*

En la figura 10, se pueden apreciar las distintas estructuras organizativas. Además de éstas, a menudo una organización compleja tiene algo de todas estas estructuras.

Figura 10: Cómo construir estructuras administrativas para proyectos (PMBOOK, 2013)



3.6 COMUNICACIONES EN UN PROYECTO

Las dimensiones posibles para la actividad de comunicación son, entre otras:

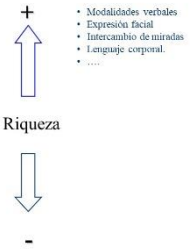
- Interna (dentro del proyecto) y externa (cliente, otros proyectos, medios de comunicación, público)
- Formal (informes, memorandos, instrucciones) e informal (correos electrónicos, conversaciones ad hoc)
- Vertical (hacia arriba y abajo dentro de la organización) y horizontal (entre colegas)
- Oficial (boletines, informe anual) y no oficial (comunicaciones extraoficiales)
- Escrita y oral
- Verbal y no verbal (inflexiones de voz, lenguaje corporal)

Para una gestión de la comunicación es necesario tomar en cuenta los flujos de información que corren por estos dos tipos de canales. Cabe señalar que las comunicaciones pueden ser transmitidas a través de diversos medios. Estos son los principales:

- Escritos: Pueden realizarse a través de comunicados, cartas, manuales, publicaciones institucionales, entre otras. Estos canales son útiles principalmente porque permiten mantener un registro tangible y verificable del mensaje a comunicar en la organización.
- Orales: Dentro de este medio, se encuentran los mensajes transmitidos durante las reuniones, las conversaciones personales y las llamadas telefónicas.
- Tecnológicos: Aunque fusiona elementos de los medios anteriores, se ha convertido en un componente muy importante dentro de la comunicación interna de las empresas. Dentro de él se encuentran los correos electrónicos, *newsletters*, el chat, las redes sociales, servicios de videollamadas, los *blogs*, etc.

Más allá de los canales utilizados para transmitir la información, es importante destacar la importancia de la implementación de un sistema de comunicación interna que permita promover una interacción bidireccional. La información debe fluir en ambas direcciones.

Medios para la comunicación



- Comunicación cara a cara.
- Medios electrónicos.
- Escritura, documentos dirigidos (cartas, memorándums, notas, y facsímiles)
- Escritura, documentos dirigidos impersonalmente (boletines, informes de computadora normales, y copias impresas)

Restricciones de los medios de comunicación

- Co-presencia: se comparte el mismo entorno físico.
- Visibilidad: visibilidad unos a otros.
- Audibilidad: se comunican hablando.
- Co-temporabilidad: el receptor recibe aproximadamente al mismo momento que el remitente produce.
- Simultaneidad: los interactuantes pueden enviar y pueden recibir al momento y simultáneamente.
- Secuencialidad: los turnos de los interactuantes no pueden salirse de una secuencia.
- Examinabilidad: los receptores pueden examinar los mensajes del remitente.
- Revisabilidad: el remitente puede revisar los mensajes antes de que él los envíe realmente.

Así, los medios de comunicación poseen las siguientes restricciones:

- Cara a cara: co-presencia, visibilidad, audibilidad, co-temporabilidad, simultaneidad, y secuencialidad.
- Medios electrónicos: audibilidad, co-temporabilidad, simultaneidad, y secuencialidad.
- Documentos escritos: examinabilidad, revisabilidad.

3.7 RIESGOS ASOCIADOS A UN PROYECTO

Concepto de riesgo

El riesgo se halla de forma implícita asociado a toda actividad:

- Todo suceso se ve marcado por las acciones del pasado, ¿Se puede, por tanto, actuar ahora para crear oportunidades en el futuro?
- El riesgo acompaña a todo cambio
- El riesgo implica elección e incertidumbre

Una posible definición es que es una variación de los resultados esperados, donde esa variación es de carácter aleatorio y en muchas ocasiones fuera del control del tomador de decisiones

generándose el problema de la incertidumbre.

Se podrían establecer 2 componentes básicos:

- Imprevisto aleatorio: que es el comportamiento o valor que pueda tomar la variable aleatoria a la que se enfrenta o bajo la cual debe trabajar y tomar decisiones un ente decisor.
- Vulnerabilidad: que es lo desprotegido o vulnerable que se encuentra, en este caso un proyecto ante uno de estos comportamientos o imprevistos aleatorios.

Las posibles estrategias o formas de afrontar el riesgo, son:

- Reactiva: Se reacciona al presentarse los problemas o imprevistos. Técnica de los bomberos.

  Método: Evaluar las consecuencias del riesgo cuando este ya se ha producido (ya no es un riesgo). Actuar en consecuencia.

  Consecuencias de una estrategia reactiva:

  - ✓ Apagado de incendios
  - ✓ Gabinetes de crisis
  - ✓ Se pone el proyecto en peligro

- Proactiva: Se busca una anticipación o predicción de los problemas o imprevistos y tener planes de contingencia.

  Método: Evaluación previa y sistemática de riesgos; Evaluación de consecuencias; Plan de evitación y minimización de consecuencias; Plan de contingencias.

  Consecuencias:

  - ✓ Evasión del riesgo
  - ✓ Menor tiempo de reacción
  - ✓ Justificación frente a los superiores

Riesgos de software

- Riesgos del proyecto. Amenaza el plan del proyecto
- Riesgos técnicos. Amenaza la calidad del producto y la planificación temporal
- Riesgos del negocio. Amenaza la viabilidad del software a ser construido (riesgos del mercado, riesgos estratégicos, riesgo dirección, riesgos del presupuesto)
- Riesgos predecibles. Predecible de la evaluación cuidadosa de plan del proyecto actual y de la experiencia de proyectos anteriores.
- Riesgos impredecibles. Algunos problemas simplemente ocurren sin advertir.

Identificación del riesgo

El tratar de identificar riesgos es un criterio proactivo que busca identificar posibles factores de riesgo y tomar medidas de aseguramiento o planes de contingencia para contrarrestarlos a ellos y a sus efectos.

Al realizar esta identificación de podría hablar de Riegos Genéricos y Riesgos específicos del producto.

- Riesgo genérico: Son todo los riesgos que afecta a cualquier proyecto. Cualquier proyecto puede presentar este tipo de problemas (tamaño del producto, impacto en el negocio, características del cliente, definición del proceso, entorno de desarrollo, tecnología a construir, experiencia del personal).
- Riesgo específico del producto: Son riesgos asociados a un producto en particular en forma única y específica. Sólo son identificables por aquellos que tienen una visión clara de la tecnología, personal y entorno específico del proyecto en cuestión.

Una metodología para identificar dichos riesgos es hacer una lista de comprobación de elementos de riesgo y para elaborarla se analizan una serie de subconjuntos de riesgos conocidos y predecibles en las siguientes subcategorías genéricas:

- Tamaño del producto: Riesgos asociados al tamaño del software, a mayor tamaño mayor riesgo.
- Impacto del negocio: Riesgos asociados a las limitaciones impuestas por la gestión o por el mercado.
- Riesgos con el cliente: Riesgos asociados a la sofisticación del cliente y la habilidad del desarrollador para comunicarse con él a tiempo. Es la definición de un buen o un mal cliente, donde por mal cliente se entiende un cliente de alto riesgo.
- Definición del proceso: Riesgos asociados al grado de definición del proceso de software y su seguimiento, como son errores de diseño y análisis.
- Entorno de desarrollo: Riesgos asociados con la disponibilidad y la calidad de herramientas que se van a emplear en la construcción del producto, calidad de herramientas de gestión, análisis, diseño.
- Riesgos tecnológicos: Riesgos implícitos en la complejidad que tendrá el sistema a construir y la tecnología de punta que contiene el sistema.
- Tamaño y experiencia del recurso humano: Riesgos asociados con la experiencia técnica y de proyectos de los ingenieros de software que van a realizar el trabajo.

Proyección del riesgo

En la proyección del riesgo se busca hacer una medición de los riesgos en función de la probabilidad de ocurrencia de los eventos y el costo o consecuencias de que ocurriese el imprevisto.

Entre el director de proyecto y su equipo pueden llevar a cabo 4 actividades para estas estimaciones:

- Establecer una escala que refleje la probabilidad percibida de riesgo.
- Definir consecuencias del riesgo.

- Estimar impacto del riesgo tanto en proyecto como en producto.
- Apuntar la exactitud general de la proyección del riesgo de manera que no haya confusiones.

Tabla de riesgos

Una técnica existente es desarrollar una tabla de riesgo para hacer la proyección. Los pasos generales serían:

1. Hacer un listado de todos los riesgos posibles sin importar lo remotos que sean.
2. Categorizar cada riesgo, indicando que tipo de riesgo es, si es con tamaño de producto, tipo de cliente, etc.
3. Determinar la probabilidad de ocurrencia de cada uno de los eventos planteados, ya sea en forma objetiva o subjetiva, y se puede hacer una valoración promediada de todas las estimaciones individuales hechas para cada evento.
4. Se valora el impacto de cada riesgo y se establece una categoría de impacto. Dichas categorías son:
    1. CATASTROFICO
    2. CRITICO
    3. MARGINAL
    4. DESPRECIABLE

5. Se realiza una ordenación en función de la probabilidad y el impacto quedando en la parte superior las de mayor probabilidad y mayor impacto.
6. Se establece una línea de corte a partir de la cual se establece que eventos serán de verdadera atención y control.
7. Se establece un plan RSGR que es un Plan de Reducción, Supervisión y gestión de riesgo.

En la tabla 20 se muestra un ejemplo de tabla riegos, con su respectiva línea de corte y entradas al RSGR.

Tabla 20: Tabla de riesgos

Riesgos	*Categoría*	*Prob.*	*Impacto*	*Rsgr*
El cliente cambiará los requisitos	Tamaño del Producto	80%	2	
Falta de formación en las herramientas	Entorno de Desarrollo	80%	3	
Menos reutilización de la prevista	Tamaño del Producto	70%	2	
La estimación del tamaño puede ser muy baja	Tamaño del Producto	60%	2	
Habrá muchos cambios de personal	Equipo de Desarrollo	60%	2	
La fecha de entrega estará muy ajustada	Riesgos del Negocio	50%	2	
Se perderán los presupuestos	Cliente	40%	1	
Línea de corte				
Los usuarios finales se resisten al sistema	Riesgos del Negocio	40%	3	
La tecnología no alcanzará las expectativas	Riesgos Tecnológicos	30%	1	
Personal sin experiencia	Equipo de Desarrollo	30%	2	
Mayor número de usuarios de los previstos	Tamaño del Producto	30%	3	

Plan de Reducción, Supervisión y gestión de riesgo

Se busca fundamentalmente, evitar el riesgo, supervisar el riesgo, gestionar el riesgo y hacer planes de contingencia

El equipo debe tener siempre una actitud proactiva frente al riesgo, evitar es la mejor estrategia, esto se logra con un plan de reducción de riesgo.

Unos pasos para esto pueden ser:

- Reunirse con el personal y determinar causas del problema
- Actuar para reducir las causas antes de comenzar proyecto
- Una vez comenzado el proyecto asegurar lo planeado como solución
- Establecer estándares de documentación y que todo se complemente a tiempo
- Convocar reuniones de revisión
- No tener irremplazables

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.

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
    Para Cada Riesgo:

    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

3.8 CALIDAD EN UN PROYECTO

Es importante mencionar que los temas de Calidad, Control de calidad y Aseguramiento de la calidad, son tratados en base al documento de la Dra. Angélica De Antonio de la Universidad Politécnica de Madrid, España, con autorización.

Concepto de la calidad

Sin un buen proceso de desarrollo es casi imposible obtener un buen producto. La calidad de un producto software debe ser considerada en todos sus estados de evolución (especificaciones, diseño, código...). No basta con tener en cuenta la calidad del producto una vez finalizado, cuando los problemas de mala calidad ya no tienen solución o la solución es muy costosa.

La calidad del producto software se diferencia de la calidad de otros productos de fabricación industrial, ya que el software tiene ciertas características especiales:

- El software es un producto mental, no restringido por las leyes de la Física o por los límites de los procesos de fabricación. Es algo abstracto y su calidad también lo es.
- Se desarrolla, no se fabrica. El costo está fundamentalmente en el proceso de diseño, no en la producción. Y los errores se introducen también en el diseño, no en la producción.
- El software no se deteriora con el tiempo. No es susceptible a los efectos del entorno, y su curva de fallos es muy diferente de la del hardware. Todos los problemas que surjan durante el mantenimiento estaban allí desde el principio y afectan a todas las copias del mismo.
- Es artesanal en gran medida. El software, en su mayoría, se construye a medida, en vez de ser construido ensamblando componentes existentes y ya probados, lo que dificulta aún más el control de su calidad.
- El mantenimiento del software es mucho más complejo que el mantenimiento del hardware. Cuando un componente hardware se deteriora se sustituye por una pieza de repuesto, pero cada fallo en el software implica un error en el diseño o en el proceso mediante el cual se tradujo el diseño en código máquina ejecutable.
- Es engañosamente fácil realizar cambios sobre un producto software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.
- Como disciplina, el desarrollo de software es aún muy joven, por lo que las técnicas de las que disponemos aún no son totalmente efectivas o no están totalmente calibradas.
- El software con errores no se rechaza. Se asume que es inevitable que el software presente errores.

Los principales problemas a los que se enfrenta a la hora de hablar de la calidad de un producto software son:

- La definición misma de la calidad del software: ¿Es realmente posible encontrar un conjunto de propiedades en un producto software que nos den una indicación de su calidad?. Para dar respuesta a estas preguntas aparecen los Modelos de Calidad.
- La calidad es un concepto que se deriva de un conjunto de sub-conceptos.
- La comprobación de la calidad del software: ¿Cómo medir el grado de calidad de un producto software? Aquí aparece el concepto de Control de Calidad, del que se hablará más adelante.

Algunas definiciones

- "Conjunto de propiedades y de características de un producto o servicio, que le confieren capacidad para satisfacer una necesidades expresadas o implícitas" (ISO 8402).
- "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).
- "Capacidad del producto software para satisfacer los requisitos establecidos" (DoD 2168).
- "Concordancia del software producido con los requerimientos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requerimientos implícitos no establecidos formalmente, que desea el usuario" (Pressman, 1998).

Las definiciones que se han dado de la calidad son demasiado generales como para que resulten de utilidad a la hora de construir un software de calidad. Por eso surge el concepto de Modelo de Calidad, que ayuda a definir la calidad del software de una forma más útil.

Modelos para la calidad

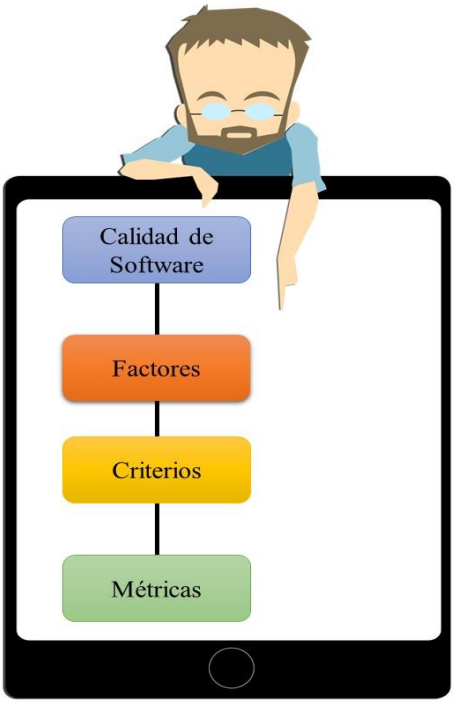
Los modelos de calidad del software vienen a ayudar en la puesta en práctica del concepto general de calidad, ofreciendo una definición más operacional. Unos de los modelos de calidad más antiguos y extendidos es el de McCall (McCall, 1977), y de él han derivado otros modelos, como el de Boehm (Boehm, 1978). En los modelos de calidad, la calidad se define de forma jerárquica. Es un concepto que se deriva de un conjunto de sub-conceptos, cada uno los cuales se van a evaluar a través de un conjunto de indicadores o métricas. Tienen una estructura, por lo general, en tres niveles, como se indica en la figura 11.

En el nivel más alto de la jerarquía se encuentran los FACTORES de calidad, que representan la calidad desde el punto de vista del usuario. Son las características que componen la calidad. También se les llama Atributos de Calidad Externos.

Cada uno de los factores se descompone en un conjunto de CRITERIOS de calidad. Son atributos que, cuando están presentes, contribuyen al aspecto de la calidad que el factor asociado representa. Se trata de una visión de la calidad desde el punto de vista del producto software. También se les llama Atributos de Calidad Internos.

Para cada uno de los criterios de calidad se definen entonces un conjunto de MÉTRICAS, que son medidas cuantitativas de ciertas características del producto que, cuando están presentes, dan una indicación del grado en que dicho producto posee un determinado atributo de calidad.

Figura 11: Estructura Jerárquica



Factores de calidad

Se centran en tres aspectos importantes de un producto software (McCall): Características operativas (operación); Capacidad de soportar los cambios (revisión) y Adaptabilidad a nuevos entornos (transición). Los más importantes se listan a continuación.

1. Corrección: Hasta qué punto un programa cumple sus especificaciones y satisface los objetivos del usuario. Es quizás el factor más importante, aunque puede no servir de nada sin los demás factores. Por ejemplo, si un programa debe ser capaz de sumar dos números y en lugar de sumar los multiplica, es un programa incorrecto.
2. Fiabilidad: Hasta qué punto se puede confiar en el funcionamiento sin errores del programa. Por ejemplo, si el programa anterior suma dos números, pero en un 25% de los casos el resultado que da no es correcto, es poco fiable.
3. Eficiencia: Cantidad de código y de recursos informáticos (CPU, memoria) que precisa un programa para desempeñar su función. Un programa que suma dos números y necesita 2 MB de memoria para funcionar, o que tarda 2 horas en dar una respuesta, es poco eficiente.
4. Integridad: Hasta qué punto se controlan los accesos ilegales a programas o datos. Un programa que permite el acceso de personas no autorizadas a ciertos datos es poco íntegro.
5. Facilidad de uso: El costo y esfuerzo de aprender a manejar un producto, preparar la entrada de datos e interpretar la salida del mismo.
6. Facilidad de mantenimiento: El costo de localizar y corregir defectos en un programa que aparecen durante su funcionamiento.
7. Facilidad de prueba: El costo de probar un programa para comprobar que satisface sus requisitos. Por ejemplo, si un programa requiere desarrollar una simulación completa de un sistema para poder probar que funciona bien, es un programa difícil de probar.

8. Flexibilidad: El costo de modificación del producto cuando cambian sus especificaciones.
9. Portabilidad: El costo de transportar o migrar un producto de una configuración hardware o entorno operativo a otro.
10. Facilidad de Reutilización: Hasta qué punto se puede transferir un módulo o programa del presente sistema a otra aplicación, y con qué esfuerzo.
11. Interoperabilidad: El costo y esfuerzo necesario para hacer que el software pueda operar conjuntamente con otros sistemas o aplicaciones software externos.

Criterios

En la tabla 21, se describen los criterios asociados a cada uno de los factores.

Tabla 21: Criterios

Factor	*Criterios*
Facilidad de uso	Facilidad de operación , Facilidad de comunicación , Facilidad de aprendizaje
Integridad	Control de accesos, Facilidad de auditoria
Corrección	Completitud, Consistencia, Trazabilidad
Fiabilidad	Precisión, Consistencia, Tolerancia a fallos, Modularidad, Simplicidad
Eficiencia	Eficiencia en ejecución, Eficiencia en almacenamiento
Facilidad de mantenimiento	Modularidad, Simplicidad, Consistencia, Concisión, Auto descripción
Facilidad de prueba	Modularidad, Simplicidad, Auto descripción, Instrumentación
Flexibilidad	Auto descripción, Capacidad de expansión, Generalidad, Modularidad
Reusabilidad	Auto descripción, Generalidad, Modularidad, Independencia entre sistema y software, Independencia del hardware
Interoperabilidad	Modularidad, Compatibilidad de comunicaciones, Compatibilidad de datos
Portabilidad	Auto descripción, Modularidad, Independencia entre sistema y software, Independencia del hardware

Métricas

En cuanto a las métricas, McCall propuso 41, sobre todo métricas de tipo subjetivo, es decir, métricas que evaluadas por personas diferentes podrían dar como resultado valores diferentes. Aún hoy en día no hay métricas formales y objetivas que cubran todos los criterios del modelo de McCall.

¿Qué es una métrica?

Aunque los términos “medida”, “medición” y “métrica” se utilizan a menudo indistintamente, es importante destacar las diferencias entre ellos, antes de establecer una definición precisa del concepto métrica.

Dentro del contexto de ingeniería de software, una “medida” proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto.

La “medición” es el acto de determinar una “medida”. Por ejemplo, una medida de dimensiones de

un producto de software son sus líneas de código (LDC), una medición de un producto particular puede señalar 10.250 LDC.

El IEEE *Standard Glossary of Software Engineering Terms* (1993), define métrica como "una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado".

Se puede definir las Métricas de Software o Medidas de Software como:

> La aplicación continua de técnicas basadas en las medidas de los procesos de desarrollo del software y sus productos, para producir una información de gestión significativa y a tiempo. Esta información se utilizará para mejorar esos procesos y los productos que se obtienen de ellos.

Esencialmente, las Métricas de Software se aplican al producto software y a los procesos mediante los que se desarrolla.

Áreas de Aplicación

La más consolidada de las áreas en el estudio de las métricas es la correspondiente a las técnicas de estimación de costos y tamaño.

¿Por qué Medir?

Es decir, ¿por qué determinar cuantitativamente una medida o métrica del software?. Esencialmente se hace para poder estimar y predecir costos, y para "conocer" la productividad y calidad del producto, evaluar la madurez de una organización, etc.

Ejemplos de métricas

- Fiabilidad = 1 - (número de errores/ número de líneas de código)
- Facilidad de mantenimiento = 1 - 0.1 (número medio de días-hombre por corrección)
- Portabilidad = 1 - (esfuerzo para portar / esfuerzo para implementar)
- Flexibilidad = 1 - 0.05 (número medio de días-hombre por cambio)
- Otras métricas se refieren al esfuerzo necesario para aprender a manejar el sistema, o la velocidad a la que trabaja el usuario una vez entrenado, o los errores que se cometen en el trabajo normal. Pero para medir lo agradable que resulta la interfaz, lo mejor es hacer encuestas a los usuarios.
- Otras métricas se refieren al proceso de mantenimiento, como: tiempo medio de reparación o cambio, número de problemas sin resolver, tiempo empleado en problemas sin resolver, porcentaje de cambios que introducen defectos, número de módulos afectados por cada cambio.
- También las métricas de complejidad se utilizan como indicadores de la facilidad de mantenimiento. La estructuración del código y de la documentación asociada suele estar muy relacionada con la facilidad de mantenimiento. Pero es más una forma de identificar potenciales puntos peligrosos que una predicción de pobre mantenibilidad.
- El concepto de funcionalidad de un producto se origina a partir de una noción intuitiva de la cantidad de funciones que proporciona. Puntos de Función (FPA, del inglés *Function Point*

Analysis). Modelo de Gustav Karner en el año 1993, basado en el método de Puntos de función. Puntos de Casos de Uso.

Métricas cuantitativas

Es difícil desarrollar medidas directas de atributos tales como: Portabilidad, Reusabilidad, etc. Por lo tanto se define un conjunto de métricas usadas para desarrollar expresiones para cada uno de los factores de acuerdo con la siguiente relación:

$$fc = c_1 * m_1 + c_2 * m_2 + ... + c_n * m_n$$

Donde:

fc = Es un factor de calidad
c_i = Coeficiente (peso) (depende de los productos en particular)
m_i = Métricas que afectan al factor

IMS

El estándar IEEE 982.1-1988, sugiere un índice de madurez del Software (IMS), que proporciona una indicación de la estabilidad de un producto de software (basada en los cambios que se producen en cada versión del producto). Se determina en base a la siguiente información:

MT = número de módulos en la versión actual
Fm = número de módulos en la versión actual que han sido modificados
Fa = número de módulos en la versión actual que han sido agregados
Fe = número de módulos de la versión anterior que se han eliminado en la versión actual

El índice de madurez del software se calcula como:

$$IMS = \frac{[MT - (Fa + Fm + Fe)]}{MT}$$

A medida que IMS se aproxima a 1; el producto comienza a estabilizarse.

Uso de un modelo de calidad

Dependiendo del grado de conformidad con el modelo de calidad seleccionado como referencia para un proyecto, se pueden adoptar dos estrategias:

Modelo predefinido

- Se aceptan los factores, criterios y métricas que propone el modelo.
- Se aceptan las relaciones entre factores y criterios, y entre criterios y métricas.
- Sólo es necesario seleccionar un subconjunto de los factores de calidad como requisitos de

calidad para el proyecto.

Definición particular de un modelo

- Se acepta la filosofía de la descomposición.
- Se selecciona un subconjunto de los factores de calidad como requisitos de calidad para el proyecto.
- Se decide la descomposición más adecuada para los factores de calidad seleccionados.

Los pasos para el uso de un modelo de calidad son:

A. Al principio del proyecto

Al especificar la calidad requerida de un producto software hay que (6 pasos):

1. Seleccionar cuáles de los factores de calidad van a ser requisitos de calidad del sistema. Para ello hay que tener varias cosas en consideración:

   La relación que tienen los factores con las características distintivas del producto o proyecto. Así, por ejemplo, si se espera que el ciclo de vida del sistema sea largo, la 'facilidad de mantenimiento' y la 'flexibilidad' se convierten en un requisito; si el sistema es experimental y se espera que las especificaciones del sistema cambien frecuentemente, la 'flexibilidad' será importante y sin embargo la 'eficiencia' apenas tendrá importancia; etc.

   El costo del factor de calidad frente al beneficio que proporciona. La tabla 22 indica, para cada factor, el ahorro que se puede esperar cuando se consigue, frente al costo necesario para alcanzar dicho factor.

Tabla 22: Beneficio frente al costo

Factor	***Beneficio Frente al costo***
Facilidad de uso	Alto
Integridad	Alto
Corrección	Bajo
Fiabilidad	Bajo
Eficiencia	Medio
Facilidad de mantenimiento	Alto
Facilidad de prueba	Alto
Flexibilidad	Medio
Reusabilidad	Medio
Interoperabilidad	Medio
Portabilidad	Bajo

   Las implicaciones de los factores de calidad sobre el ciclo de vida, es decir, en qué etapas es necesario evaluar cada uno de los factores de calidad, y en qué etapas se dejan sentir los efectos de una calidad pobre con respecto a cada uno de estos factores.

Las interrelaciones entre factores. Algunos factores pueden ser conflictivos entre sí. La eficiencia, por ejemplo, está en conflicto con prácticamente todos los demás factores de calidad. La siguiente tabla indica la dependencia entre los factores de McCall.

2. Una vez seleccionados los factores de calidad que son requisitos para el producto, es necesario organizarlos en orden de importancia.
3. Una vez establecidos los factores de calidad, el modelo de calidad proporciona automáticamente el conjunto de atributos o criterios relacionados con dichos factores.
4. Para cada uno de los criterios de calidad se definen o eligen un conjunto de métricas.
5. Se debe establecer valores deseables para los criterios en función de datos históricos, el promedio en la industria, etc. Se pueden establecer valores finales, es decir, los que se desea obtener una vez finalizado el desarrollo, y también valores intermedios o predictivos en cada período de medición durante el desarrollo.
6. Por último, se deberán establecer los valores mínimos aceptables.

B. Durante el desarrollo

Hay que (3 pasos):

1. Implementar las métricas, es decir, tomar las medidas necesarias.
2. Analizar los resultados de las métricas.
3. Tomar acciones correctivas si es necesario, es decir, si los valores obtenidos están por debajo de los valores mínimos aceptables. Estas medidas correctivas pueden afectar tanto al proceso de desarrollo como al proceso de gestión.

C. Al final del proyecto

Será necesario:

Validar las medidas predictivas utilizadas, y comprobar si en efecto se pueden tomar como indicadores de los valores finales.

LECTURA ADICIONAL RECOMENDADA

1. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.
2. Pressman, Roger s., Ingeniería de software, un enfoque práctico, sexta edición. Editorial McGraw Hill, México, año 2006.
3. Sommerville, Ian, Ingeniería de software, sétima edición, Editorial Pearson Addison Wesley, España, año 2005.
4. De Antonio, A., Gestión, control y garantía de la calidad del software, Apuntes, Universidad Politécnica de Madrid, año 2004.
5. Boehm, B.W., Brown, J.R., Lipow, M., Macleod, G.J., Merritt, M.J.; Characteristics of Software Quality, North-Holland, 1978.
6. J.A. McCall, P.K. Richards,.G.F. Walters, Factors in Software Quality, RADC-TR-77-369, Rome Air Development Center, United States Air Force, 1977.

EJERCICIOS PROPUESTOS

3.1 Encuentre la ruta crítica de los siguientes proyectos

a. El tiempo de duración de cada actividad en semanas es fijo. Se solicita que estime la duración total del proyecto a través del método CPM.

Actividad	Duración (sem)	Actividad Predecesora
A	6	-
B	8	-
C	12	A,B
D	4	C
E	6	C
F	15	D,E
G	12	E
H	8	F,G

b. Un proyecto que consta de **9 actividades** cuyos tiempos estimados se encuentran en semanas. Adicionalmente en la columna ***"Predecesor"*** se establece el orden en el cual se deben realizar las distintas actividades, por ejemplo, la **Actividad G** se puede realizar una vez completada las **Actividades D** y **F**.

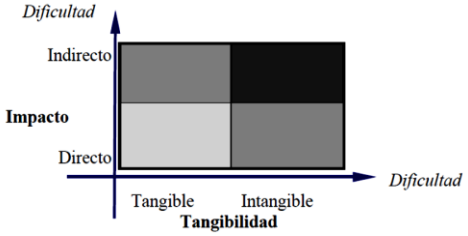
Actividad	Predecesor	Duración [sem]
A	-	5
B	-	6
C	A	4
D	A	3
E	A	1
F	E	4
G	D,F	14
H	B	12
I	H,C,G	2

c. Para el siguiente cronograma de actividades

Actividad	Actividad Predecesora	Tiempo
A	---	3h
B	A	1h
C	A	2h
D	A	2h
E	B-C-D	4h

3.2 Costos y Beneficios asociados a Proyectos Informáticos

a. Explique la siguiente figura, ejemplificando para cada tipo costo y beneficio



b. Para su proyecto, liste en una tabla los costos y beneficios que traerá su proyecto. Ejemplifique y explique brevemente cada uno de ellos.

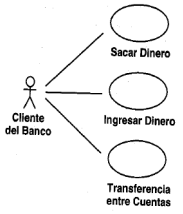
3.3 Estimación de Costos

a. Qué es la estimación de costos en el desarrollo de software
b. Qué es proceso de estimación de costos
c. La estimación de costos y la planificación se hacen independientemente. Justifique
d. Qué es la productividad. Ejemplifique
e. Qué factores afectan la productividad. Ejemplifique
f. Explique las técnicas para la estimación de costos
g. Cuáles son las dos partes de la técnica algorítmica para estimar costos
h. Indique el número de transacciones en cada uno de los siguientes casos de uso

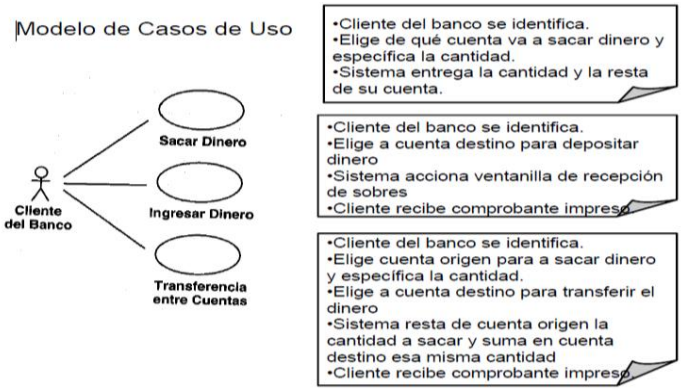
1. Indique cuantas Transacciones hay en esta especificación preliminar de portal Web (IGU):

(1) The user selects an X.
(2) The system shows the Y s connected with this X.
(3) The user also selects one or more Y s and submits.
(4) The system searches for hits and shows the results.

2. Un sistema de Control de Cajero Automático, tiene, entre otros, los siguientes casos de uso:



En una primera aproximación para estimar costos se visualizaron los siguientes detalles para los casos de uso. Indique cuantas transacciones en total hay en los tres casos de uso.



3. En una segunda revisión, por parte de un equipo más experimentado, se visualizan las siguientes especificaciones para el caso de uso "Sacar Dinero"

   Curso típico de eventos.

   Cliente introduce su tarjeta en el cajero.
   El sistema pide la clave de identificación.
   El cliente introduce la clave.
   El sistema presenta las opciones disponibles.
   El cliente selecciona la operación de retiro.
   El sistema pide la cantidad a retirar.
   El cliente introduce la cantidad requerida.
   El sistema procesa la petición y eventualmente, da el dinero solicitado.
   El sistema devuelve la tarjeta y genera un recibo.
   El cliente recoge el dinero, el recibo y la tarjeta. Procede a retirarse.

4. Caso de uso "**Buscar un contacto**"

   1. El usuario introduce el texto y otros criterios de búsqueda.
   2. El sistema realiza la búsqueda y muestra los resultados.
   3. El usuario pulsa sobre un contacto.
   4. El sistema muestra los datos del contacto.

5. Caso de Uso: "**Generar la Certificación Manual"**

   1. El usuario del Sistema Cobranza entra al sistema ingresando el Usuario y Contraseña.
   2. El sistema muestra la pantalla principal que incluye un menú de opciones.
   3. El usuario del Sistema selecciona la opción de "Certificación Manual".
   4. El sistema muestra una pantalla donde pregunta al usuario si hubo o no cheques devueltos.
   5. El usuario selecciona que no hubo cheques devueltos.
   6. El sistema muestra una serie de filtros para buscar liquidaciones en el sistema.

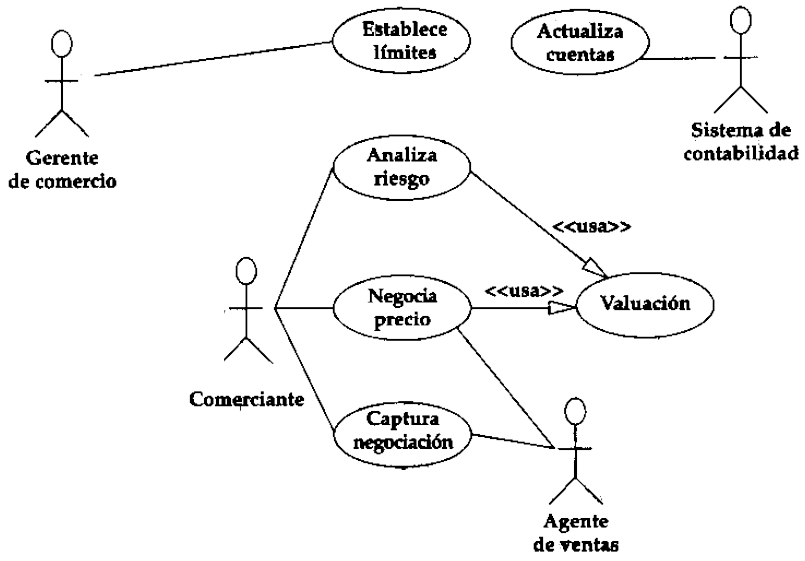
7. El usuario selecciona y captura los filtros de acuerdo a las formas de pago que desea buscar.
8. El sistema muestra un listado con las liquidaciones que coinciden con los filtros seleccionados por el usuario.
9. El usuario presiona la liga para mostrar el detalle de la liquidación.
10. El sistema muestra el detalle de las formas de pago de la liquidación.
11. El usuario selecciona las formas de pago a certificar.
12. El usuario presiona el botón para certificar las formas de pago seleccionadas.
13. El Sistema certifica los pagos seleccionados por el usuario.

6. Caso de Uso: Consultar licenciamiento de contenido:

1. El usuario selecciona el contenido del que desea ver las restricciones
2. El usuario indica al sistema que desea revisar cuáles son las restricciones aplicadas a dicho contenido
3. El sistema muestra al usuario las restricciones aplicadas a dicho contenido

i. Aplicando el método de Karner, estime el costo de desarrollo de los siguientes proyectos

1. El subsistema autónomo en proyecto, es un gestor de "créditos" y se ha derivado el siguiente Diagrama de Casos de Uso:



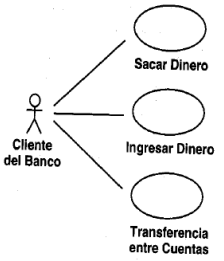
Los diseñadores, prevén que cada uno de los casos de uso consistirá de 8 transacciones. También la interacción humana con el sistema será a través de una IGU.

En un análisis en mayor profundidad, se han determinado que todos los factores de complejidad técnico (T_1 a T_{13}) para construir el subsistema son de nivel "esencial"

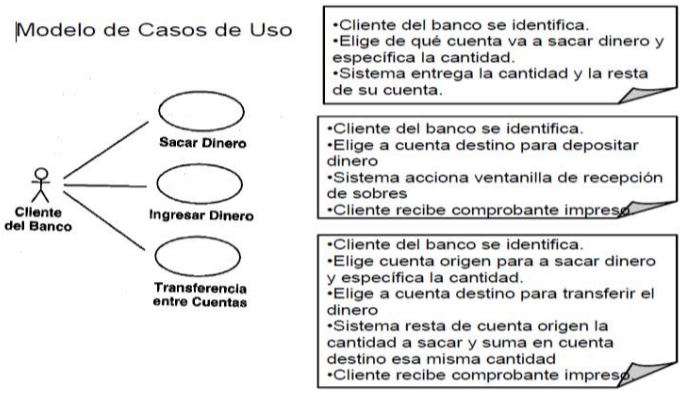
y así mismo, en el análisis, se han determinado que todos factores de complejidad ambientales (E_1 a E_8) para construir el subsistema son de nivel "promedio".

Usted sabe que en su empresa cada Punto de Casos de Uso requiere 22,5 horas-hombre.

2. Un sistema de Control de Cajero Automático, tiene, entre otros, los siguientes casos de uso:



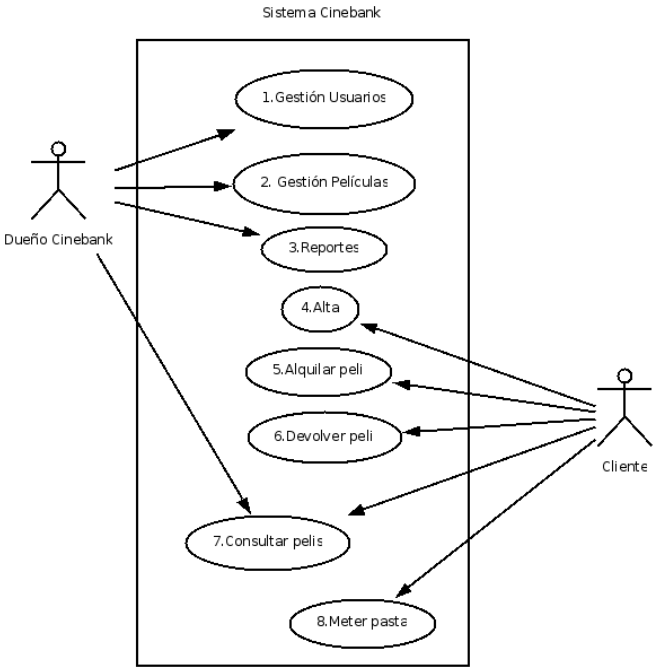
En una primera aproximación para estimar costos se visualizaron los siguientes detalles para los casos de uso.



En un análisis en mayor profundidad, se han determinado que todos los factores de complejidad técnico (T_1 a T_{13}) para construir el subsistema son de nivel "esencial" y así mismo, en el análisis, se han determinado que todos factores de complejidad ambientales (E_1 a E_8) para construir el subsistema son de nivel "Amplia experiencia".

Usted sabe que en su empresa cada Punto de Casos de Uso requiere 22,5 horas-hombre.

3. Un gestor de "Alquiler de Películas" y se ha derivado el siguiente Diagrama de Casos de Uso:



Los diseñadores, prevén que cada uno de los casos de uso consistirá de 3 transacciones. Así mismo, tanto el Dueño como los clientes son usuarios profesionales con muchos años de experiencia y tienen un buen nivel de uso de la computadora.

En un análisis en mayor profundidad, se han determinado que todos los factores de complejidad técnico (T_1 a T_{13}) para construir el subsistema son de nivel "esencial" y así mismo, en el análisis, se han determinado que todos los factores de complejidad ambientales (E_1 a E_8) para construir el subsistema son de nivel "promedio".

Finalmente, aunque se equipo sabe que Karner originalmente sugirió que cada Punto de Casos de Uso requiere 20 horas-hombre, usted sabe que en su empresa cada Punto de Casos de Uso requiere 22,5 horas-hombre.

3.4 Organización del Proyecto

a. Indique cuáles son las distintas formas de organizar una empresa entorno a proyectos.
b. Explique la tabla 19 de la sección 3.5.
c. Explique su propuesta de organización para el proyecto que usted dirige.

3.5 Gestión de los Riesgos

a. De una definición del riesgo
b. Señale los componentes básicos del riego
c. Indique las estrategias o formas de afrontar el riesgo
d. Señale cuáles son los riesgos asociados con el desarrollo de Software
e. Cómo se identifica el riesgo
f. Indique las estrategias de manejo de riesgos
g. Explique que es un Plan RSGR e indique qué estructura tiene el documento

h. Construya una tabla de riesgos para su proyecto
i. Construya un Plan RSGR para su proyecto

3.6 Comunicaciones

a. Construya una tabla con todos los mecanismos (dando una breve explicación) que puede aplicar para soportar la comunicación en un proyecto informático.

b. Con respecto a la Gestión de Comunicaciones en un proyecto:

   a) Una alternativa para disminuir la cantidad de enlaces de comunicación (o canales) en un grupo consiste en tener un coordinador central por grupo que gestione y centralice las comunicaciones.
   b) a) y cuanto menor es el grupo, mayor es el número de enlaces de comunicación que existen entre sus miembros.
   c) b) y siendo n la cantidad de miembros en un grupo, existen n*(n-1)/2 líneas de comunicación.
   d) c) y el ambiente físico de trabajo (luminosidad, ruido, etc.) no es un factor que afecte la comunicación en un grupo.

3.7 Calidad

Conceptos

i. ¿Por qué sin un buen proceso de desarrollo es casi imposible obtener un buen producto?.
ii. ¿Qué características especiales del software hace que la calidad del producto software se diferencia de la calidad de otros productos de fabricación industrial?.
iii. Explique con ejemplo el por qué, es engañosamente fácil realizar cambios sobre un producto software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.
iv. De una definición formal para la Calidad del Software.

Modelo de Calidad

i. Explique los tres niveles de un modelo de calidad, como el propuesto por McCall, dando ejemplos.
ii. Indique algunos criterios para los respectivos factores de calidad

FACTOR	CRITERIOS
Facilidad de uso	
Integridad	
Corrección	
Fiabilidad	
Eficiencia	
Facilidad de mantenimiento	
Facilidad de prueba	

Flexibilidad	
Reusabilidad	
Interoperabilidad	
Portabilidad	

iii. Explique los pasos para el uso de un modelo de calidad

Métricas de Calidad

i. Qué es una métrica según el *IEEE Standard Glossary of Software Engineering Terms* (1993).
ii. Métrica para medir la Portabilidad. Debe explicar como "funciona" y ejemplificar con algunos datos ficticios.

Portabilidad = 1 - (esfuerzo para portar / esfuerzo para implementar)

iii. Otras métricas que requieren ser explicadas son:

- Facilidad de mantenimiento = 1 - 0.1 (número medio de días-hombre por corrección)
- Flexibilidad = 1 - 0.05 (número medio de días-hombre por cambio)
- PCU (Puntos por Caso de Uso) = Modelo de Costos de Gustav Karner
- Cohesión en el Diseño
- Acoplamiento en el Diseño
- Líneas de Código (LdC)
- Mantenimiento del Software = Tiempo medio de reparación o cambio, Número de problemas sin resolver, Tiempo empleado en problemas sin resolver, Porcentaje de cambios que introducen defectos, Número de módulos afectados por cada cambio

iv. Calcule el IMS para un software que va en su décima versión, con un

$M_T = 1.500$
$F_m = 50$
$F_a = 10$
$F_e = 15$

¿Se está estabilizando el software en desarrollo con su décima versión?

TRABAJO DE PROYECTO

Aplicación práctica 3

Se solicita, desarrollar un informe de Planificación (Parte I) de su proyecto que incluya las siguientes secciones del capítulo III.

3.1 Plan temporal

3.1.1 Carta Gantt del proyecto

3.1.2 Red del proyecto

3.2 Estimación de Costos
- 3.2.1 Costos Asociados al Proyecto (cuantificar en MM$)
- 3.2.2 Beneficios Asociados al Proyecto (cuantificar en MM$)
- 3.2.3 Costo de la etapa desarrollo del software (modelo de Karner, cuantificando en MM$)
- 3.2.4 Tabla de Costos vs Beneficios proyectado en semanas o meses (según Plan temporal)

3.3 Diagrama contextualizado de la organización proyecto y Organización del desarrollo del software (proyectizada y su justificación)
- 3.3.1 Diagrama organizativo del proyecto
- 3.3.2 Diagrama organizativo del desarrollo del software

3.4 Mecanismos oficiales de comunicación y sus responsables (electrónicos, reuniones formales, cartas, memos, etc.)
- 3.4.1 Mecanismo 1 (justificar)
- 3.4.2 Mecanismo 2 (justificar)
- 3.4.3 …. Etc.

3.5 Riesgos

3.5.1 Tabla de riesgo del proyecto
Descripción de todos los riesgos por encima de la línea de Corte
Factores que influyen en la probabilidad e impacto

3.5.2 Reducción, supervisión y gestión del riesgo

Para Cada Riesgo:

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.

Aplicación práctica 4

Luego de haber desarrollado una primera etapa de planificación que consideró una organización para el recurso humano que desarrollará el software (proyectizada y su justificación), los mecanismos oficiales que se usarán para la comunicación de resultados del proceso (electrónicos, reuniones formales, cartas, memos, etc.), un calendario de actividades (CMP y una Gantt), una estimación de costos para construir el software (modelo de Gustav Karner) y finalmente un plan para gestión de los riesgos, se requiere desarrollar un Plan de Gestión de la Calidad.

Por lo anterior, se requiere la elaboración de un informe según el siguiente formato (sólo la sección de desarrollo, ya que el resto está determinado).

III. DESARROLLO

Las secciones del estándar IEEE 730-2002.

AYUDA:

Para desarrollar el PGC, deberá:

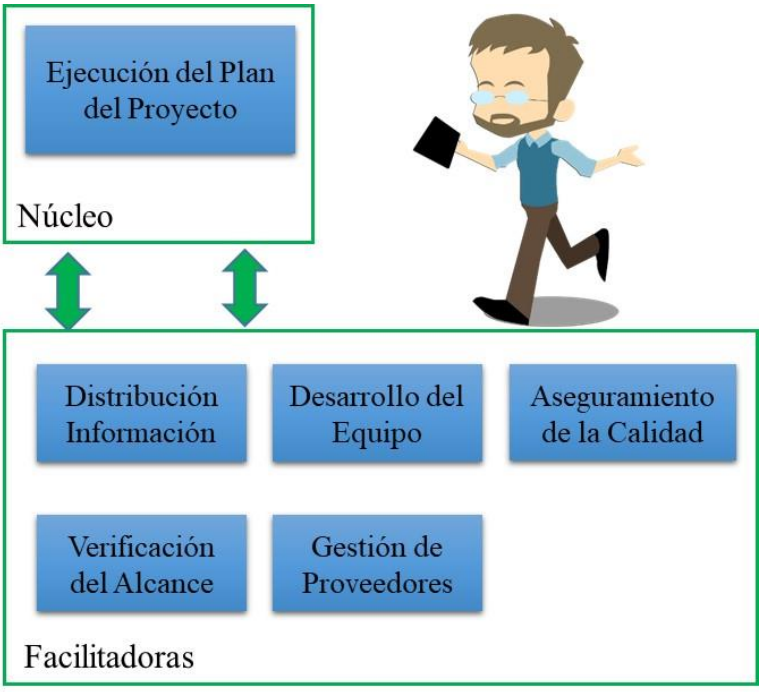
i. Analizar qué es un PGC
ii. Saber cómo se desarrolla un PGC
iii. Saber la estructura de un PGC según el estándar IEEE 730-2002
iv. Desarrollar el PGC para el proyecto definido, haciendo todas las suposiciones necesarias
v. Usar los ejemplos entregados por el profesor

CAPÍTULO 4: EJECUTAR UN PROYECTO

4.1 PROCESO DE EJECUCIÓN

En la figura 12, se muestra las tareas asociadas con el proceso de ejecución de un proyecto.

Figura 12: Tareas del proceso de ejecución



Tarea núcleo

- Ejecución del plan de proyecto (las etapas de desarrollo de software).

Tareas facilitadoras

- Verificación del alcance: formalizar aceptación del alcance (logrado) del proyecto (requisitos del proceso y producto).
- Aseguramiento de la calidad: evaluar el desempeño general para proporcionar confianza de que los estándares de calidad se van a cumplir.
- Desarrollo del equipo: desarrollar las habilidades individuales y del grupo para mejorar el desempeño del proyecto.
- Distribución de información: hacer que la información necesaria llegue a las partes interesadas de forma oportuna.
- Gestión de proveedores: recepción de solicitudes (obtener cotizaciones, propuestas y ofertas de acuerdo a lo apropiado), selección de fuentes (elegir entre los vendedores potenciales), administración del contrato (gestionar las relaciones con el proveedor).

4.2 ASEGURAMIENTO DE LA CALIDAD

¿Qué es la garantía de calidad?

Un conjunto de acciones de planificación, estimación y supervisión de las actividades de desarrollo que se realizan de forma independiente al equipo de desarrollo, de tal forma que los productos software resultante cumplen los requisitos establecidos.

También es un conjunto de procedimientos, técnicas y herramientas, aplicados por profesionales, durante el ciclo de desarrollo de un producto, para asegurar que el producto satisface o excede los estándares o niveles de calidad preestablecidos. Abarca todas aquellas actividades o prácticas que se realizan con el objetivo de asegurar un cierto nivel de calidad en el producto desarrollado. Por lo general, el equipo de Garantía de Calidad es diferente del equipo de desarrollo, especialmente en proyectos grandes.

Las áreas que caen bajo la responsabilidad del grupo de Garantía de Calidad son tres:

- Las metas y objetivos: Debe asegurar que las metas de la organización en primer lugar, y los objetivos del usuario en segundo lugar se están satisfaciendo y que no existen conflictos entre ellos o entre los objetivos de diferentes usuarios.
- Los métodos: Debe asegurar que las actividades de desarrollo de software siguen los procedimientos establecidos, se ajustan a los estándares seleccionados, están de acuerdo con las políticas de la organización y se ejecutan según las guías de trabajo y recomendaciones disponibles.
- Rendimiento: Debe asegurar que se optimiza la utilización del hardware y software en los productos desarrollados, que son económicos (se desarrollan con el menor costo posible), eficientes (sacan el máximo partido posible a los recursos utilizados) y efectivos (alcanzan el resultado deseado con la menor cantidad posible de recursos, tiempo y esfuerzo).

Las principales tareas del grupo de garantía de calidad, por lo tanto, son:

- Planificación de la calidad: Consiste en seleccionar, clasificar y ponderar las propiedades de calidad que se van a establecer como requisitos, con respecto al producto y con respecto al proceso. Se elegirán también los mecanismos de control de calidad a utilizar para medir y evaluar estas características y se determinarán las metas a alcanzar.
- Supervisión de la calidad: Consiste en supervisar y corregir, si es necesario, el trabajo que se está realizando (según los resultados obtenidos con las actividades de control de calidad), con el objetivo de llegar a satisfacer los requisitos establecidos.
- Construcción de la calidad: Actividades constructivas son aquellas que sirven para "construir" la calidad, es decir, son actividades preventivas cuyo objetivo es evitar la introducción de errores mediante la puesta en práctica de ciertos principios, métodos, formalismos y herramientas.

Se pueden considerar diferentes tipos de actividades constructivas de Garantía de Calidad:

- Técnicas: Incluyen, por ejemplo, la aplicación de principios, técnicas y herramientas de

Ingeniería de Software.

- Organizativas: Incluyen, por ejemplo, la aplicación de modelos de proceso o planes.
- Humanas: Incluyen, por ejemplo, la formación del personal y la motivación.

Los grupos afectados por la Garantía de Calidad, son:

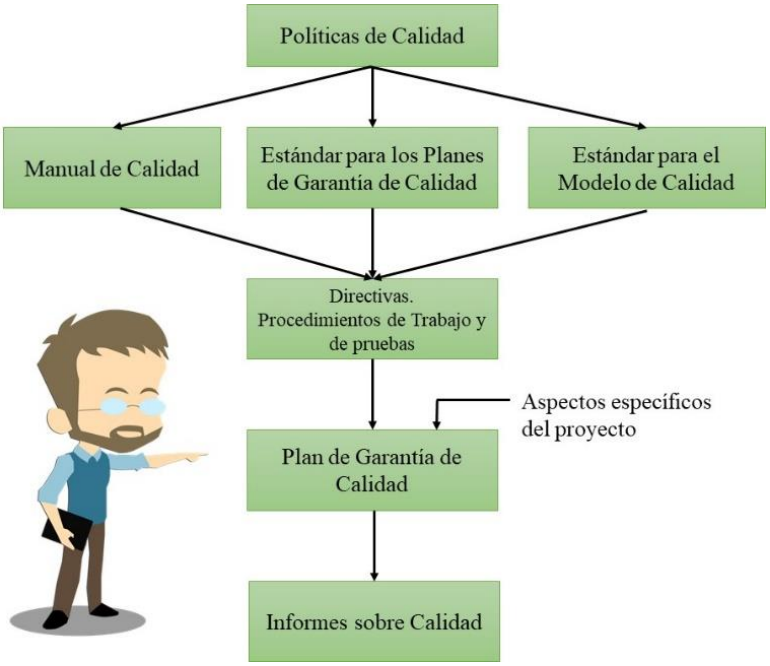
- Los usuarios, ya sean externos o internos a la organización que desarrolla el software.
- Los desarrolladores. El programa de Garantía de Calidad les ofrece un marco de trabajo estándar y estable de un proyecto a otro, sobre el que basar las responsabilidades.
- El público en general, ya que se verá afectado por el buen o mal funcionamiento de los sistemas software desarrollados.

Plan de garantía de calidad (PGC)

Un PGC es un documento que describe como la organización asegurará que se realizará el trabajo de desarrollo, operación o mantenimiento de software conforme a los procedimientos y métodos establecidos para el proyecto.

En la figura 13, se describe el proceso para escribir un PGC.

Figura 13: Proceso para escribir un PGC



Donde:

- El Manual de Calidad debe ser una guía al Sistema de Calidad. Debe especificar la terminología, políticas, principios, responsabilidades y procesos del sistema, así como los estándares en los que se basa (por ejemplo, ISO 9001, 9002 o 9003). Debe dar respuesta a las cuestiones de Quién, Dónde y Por qué la Garantía de Calidad.

- Es también necesario adoptar un estándar para la elaboración de Planes de Garantía de Calidad, como por ejemplo el IEEE Std 730-1989; 730 – 1995; 730-2002.
- Es conveniente, así mismo, basar la Garantía de Calidad en un Modelo de Calidad reconocido, que ayude en la selección de propiedades y métricas de calidad a utilizar.
- Las directivas y los procedimientos de trabajo y de prueba van a facilitar la selección y la puesta en práctica de actividades de garantía de calidad. Vienen a contestar la cuestión del Qué y el Cómo aplicar la Garantía de Calidad.
- Para cada proyecto será necesario desarrollar un Plan de Garantía de Calidad, de acuerdo con el estándar elegido y teniendo en cuenta los aspectos específicos del proyecto que van a influir en la calidad.
- El Plan de Garantía de Calidad va a describir en detalle qué actividades de Garantía de Calidad aplicar en cada una de las fases del desarrollo.
- Finalmente, los Informes de Calidad contendrán los resultados obtenidos con la aplicación de las diferentes Actividades de Garantía de Calidad.

Manual de calidad

Formaliza y concretiza la política de la empresa relativa a la Gestión de Calidad. Define los requisitos generales que deben ser establecidos en la empresa para garantizar la implantación del Sistema de Calidad y su cumplimiento. Detalla los criterios a seguir y hace referencia a los procedimientos que componen el Sistema de Calidad de la Empresa.

La estructura del Manual de Calidad, según el estándar ISO 9004-2, debe tener los siguientes apartados:

1. Generalidades
2. Organización
3. Revisión del contrato
4. Control del proyecto
5. Control de la documentación
6. Control de las compras
7. Productos o servicios suministrados por el cliente
8. Identificación y trazabilidad
9. Control de los procesos
10. Inspección y Pruebas
11. Verificación de los equipos de inspección
12. Estados de inspección
13. Control de productos o servicios no conformes
14. Acciones correctivas
15. Manipulación, almacenamiento, embalado y entrega
16. Registros sobre la calidad
17. Auditorías internas
18. Formación y adiestramiento
19. Mantenimiento del producto o servicio
20. Técnicas estadísticas

Guía de IEEE para planificar la garantía de calidad

Esta guía viene a complementar el estándar de IEEE para los Planes de Garantía de Calidad y recoge el consenso alcanzado entre un cierto número de personas experimentadas en la generación, implementación, evaluación y modificación de Planes de Garantía de Calidad de software sobre lo que son buenas prácticas de Garantía de Calidad. Es, por tanto, un conjunto de recomendaciones, no un estándar.

Tipo de software considerado: El estándar de IEEE está dirigido al desarrollo de software crítico, es decir, aquel cuyo fallo puede producir grandes pérdidas o catástrofes. Si se está desarrollando software no crítico, no tiene sentido imponer todos los requisitos del estándar.

Estructura de un PGC

1. Propósito
2. Documentos de referencia
3. Gestión
4. Documentación
5. Estándares, prácticas y convenciones
6. Revisiones y auditorias
7. Gestión de Configuración
8. Gestión de Problemas y Acciones Correctivas
9. Herramientas, técnicas y métodos
10. Control del Código
11. Control de Medios
12. Control de Suministradores y Subcontratas
13. Recolección, Mantenimiento y Retención de Registros

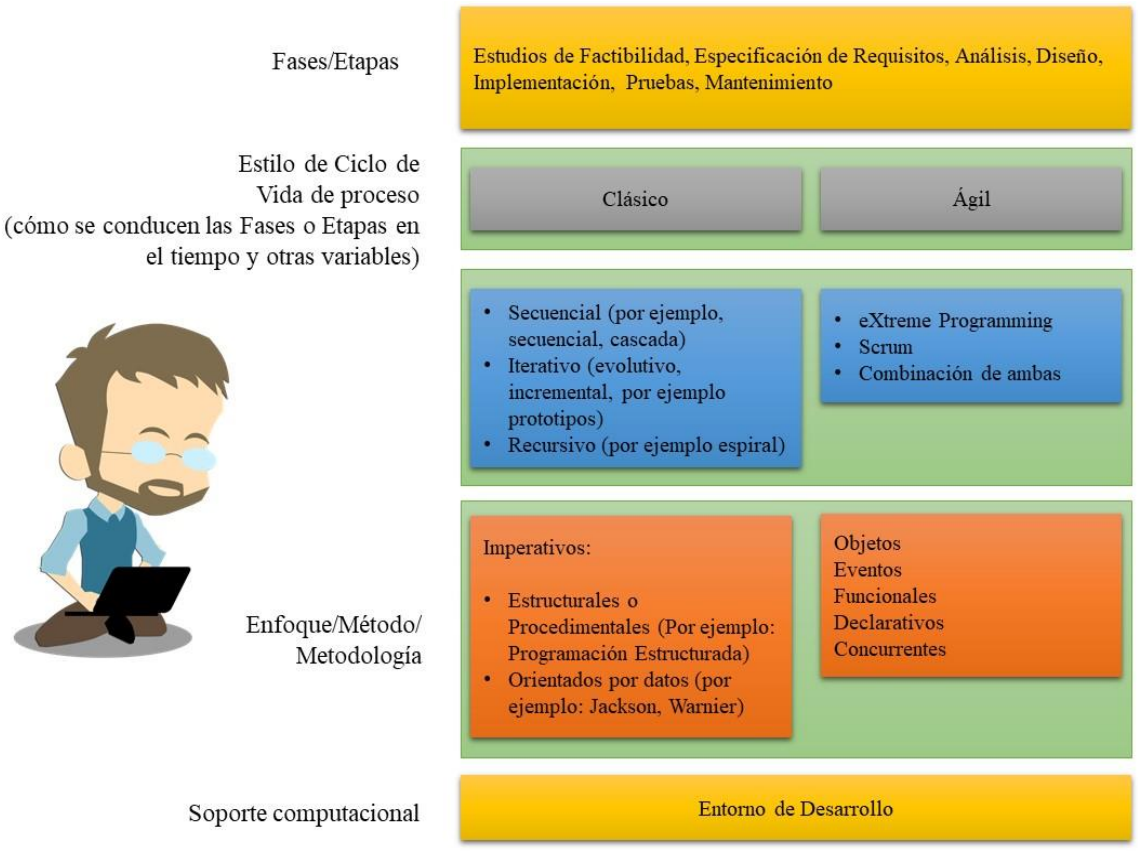
4.3 FUNDAMENTOS PARA EJECUTAR EL PROYECTO

Existe una relación entre las Fases/Etapas, Estilo de Ciclo de Vida de proceso, Enfoque/Método, y el Soporte para desarrollar un software.

En primer lugar, es necesario un conjunto de etapas o fases para conducir el desarrollo, tales como la especificación de requisitos, el análisis, el diseño, la implementación, las pruebas y el mantenimiento. Estas etapas o fases serán descritas más adelante, en esta sección. Posteriormente, las etapas deben ser conducidas en el tiempo, espacio, etc. Acá existen dos estrategias: Clásica y Ágil. En el primer caso el énfasis esta dado en el proceso, sus entradas y salidas (de cada etapa). También, en general, el producto se construye de forma progresiva, desde una idea hasta lograr el producto final requerido. Típicamente, se encuentran en una conducción secuencial, iterativa o recursiva de las etapas de desarrollo. Por su parte, en el segundo caso, el énfasis esta dado en las personas que ejecutan las etapas y en ciclos cortos determinados de tiempo. En cada ciclo, se construye un subproducto o producto en versión preliminar operativo que puede ser utilizado por el cliente. En general, la conducción de las etapas tiene un enfoque iterativo o evolutivo. En segundo lugar, independiente del estilo de conducción de las etapas o fases, es necesario establecer un enfoque de modelado o método. En este caso puede ser orientado a procesos u objetos.

Finalmente, están las herramientas de soporte que normalmente se adscriben a un enfoque de modelado, estilo de ciclo de vida y un conjunto de etapas de desarrollo. En esta sección se describirán en detalle estas ideas. En la figura 14, se presenta la relación entre estos elementos.

Figura 14: Fases/Etapas, Estilo de Ciclo de Vida de proceso, Enfoque/Método, y el Soporte para desarrollar un software

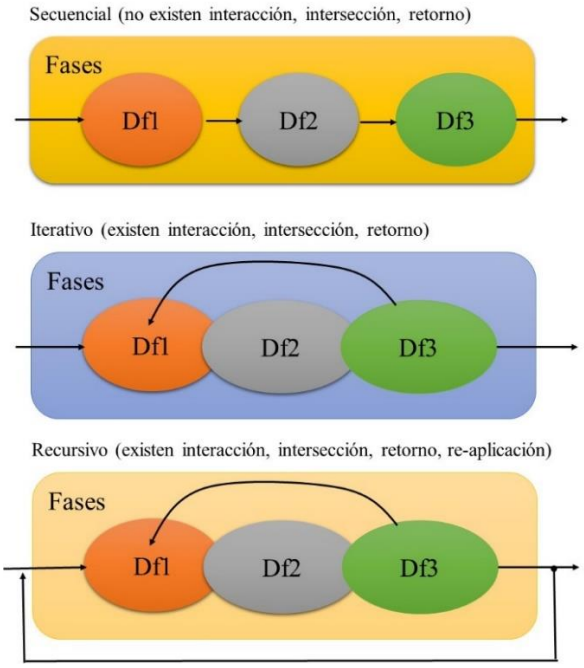


Ciclo de Vida

En la primera parte de este documento se presentó el concepto de proceso de software como un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software. Unas de las tecnologías más importantes son los ciclos de vida.

En la figura 15, se presenta una mirada genérica de los tipos de ciclo de vida existente. Posteriormente, se discutirá independientemente cada una de ellas. El primer tipo de ciclo de vida, las etapas son conducidas secuencialmente no existiendo entre ellas interacción, intersección ni retornos a etapas anteriores. El segundo es el iterativo, donde si existe interacción, es decir, las etapas pueden, por ejemplo, coexistir; también existe intersección, es decir, algunas tareas pueden, por ejemplo, ser abordadas en etapas contiguas; y existe la posibilidad de retornar a etapas ya concluidas o aún en ejecución. Finalmente, en el recursivo, todas las etapas pueden ser re-ejecutadas de manera recurrente.

Figura 15: Ciclos de vida, una mirada genérica



¿Qué es un ciclo de vida?

La ingeniería del software establece y se vale de una serie de modelos que muestran las distintas etapas y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta la retirada del producto. A estos modelos se les denomina “Modelos de ciclo de vida del software”. Describen las fases del ciclo de software y el orden en que se ejecutan las fases. Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software, intenta determinar el orden de las etapas involucradas y los criterios de transición asociados entre estas etapas.

En cada una de las etapas de un modelo de ciclo de vida, se pueden establecer una serie de objetivos, tareas y actividades que lo caracterizan. Existen distintos modelos de ciclo de vida y la elección de un modelo para un determinado tipo de proyecto es realmente importante; el orden es uno de estos puntos importantes. A continuación, se muestran algunos de los modelos tradicionales y más utilizados.

Ciclos de vida tradicionales

En la tabla 23, se describen los distintos tipos de ciclo de vida que pueden emplearse en el desarrollo de un software, bajo un enfoque tradicional.

Tabla 23: Ciclos de vida tradicionales

Estilo de Ciclo de Vida	***Descripción***	***Fortalezas***	***Debilidades***	***Dominio aplicación***
Secuencial • Lineal • Cascada • Cascada Sashimi • Cascada con Subproyectos Ver figura 16.	El proyecto se descompone en etapas separadas que son realizadas de manera lineal y secuencial. Enfatiza el cumplimiento de una fase del desarrollo antes de pasar a la fase siguiente. Las actividades de cada etapa son independientes entre sí. No hay retroalimentación entre ellas. Congela los productos de una fase antes de pasar a la siguiente Cada actividad genera entradas y documentación para la siguiente.	Es el más sencillo de todos. Menos costos en gestión.	Proyectos reales raras veces se ajustan. Raras veces el cliente expone todos los requisitos al inicio. No existe retroalimentación, costoso ante cambios. Errores detectados tardíamente. Producto operativo al final. Paciencia (cliente) alta. Todo o nada.	Cuando todos los requerimientos han sido establecidos claramente al inicio o la organización está familiarizada con el dominio. En proyectos pequeños y simples.
Iterativo • Incremental • Evolutivo • Prototipos Ver figura 17.	Es básicamente el mismo proceso que se realiza en el modelo cascada, pero desarrollándose en secciones que se entrecruzan Se basa en la filosofía de construir incrementando las funcionalidades del programa. Se realiza construyendo por módulos que cumplen las diferentes funciones del sistema. Aumenta gradualmente las funcionalidades. Es una repetición del ciclo de vida en cascada en cada funcionalidad. Reduce los riesgos ya que va construyendo partes del sistema adoptando este modelo.	Los requerimientos no necesitan ser totalmente especificados/clarificados desde el principio Para cada incremento que se completa, los requerimientos se clarifican	Existe una tendencia a dejar los problemas complejos para el futuro de manera de demostrar éxito temprano en el manejo Dificultad para manejar y medir el proyecto porque no se puede determinar cuando todos los requerimientos serán completados	Proyectos donde los requerimientos no puedan ser bien especificados
Espiral Ver figura 18.	Divide el desarrollo del sistema en cuatro actividades básicas: • Planificación • Análisis de riesgos • Ingeniería • Evaluación Con cada vuelta del espiral, los riesgos son identificados y se ejecutan intentos de mitigar los riesgos antes de proceder a las actividades de ingeniería del espiral.	Evita algunas de las dificultades existentes en los modelos de software por el uso de la aproximación guiada por riesgos. Trata de eliminar los errores en fases tempranas. Existe reevaluación después de cada fase, permitiendo cambios en las perspectivas del usuario, tecnología, avances o perspectivas financieras.	Provee más flexibilidad de la conveniente para muchas aplicaciones. Necesita experiencia en la determinación de riesgos, y dado que esta no es una tarea sencilla, para realizar una esta tarea, es necesaria una gran experiencia en proyectos para realizarla de manera exitosa.	Funciona bien para proyectos complejos, dinámicos e innovadores.

Figura 16: Ciclos de vida secuencial

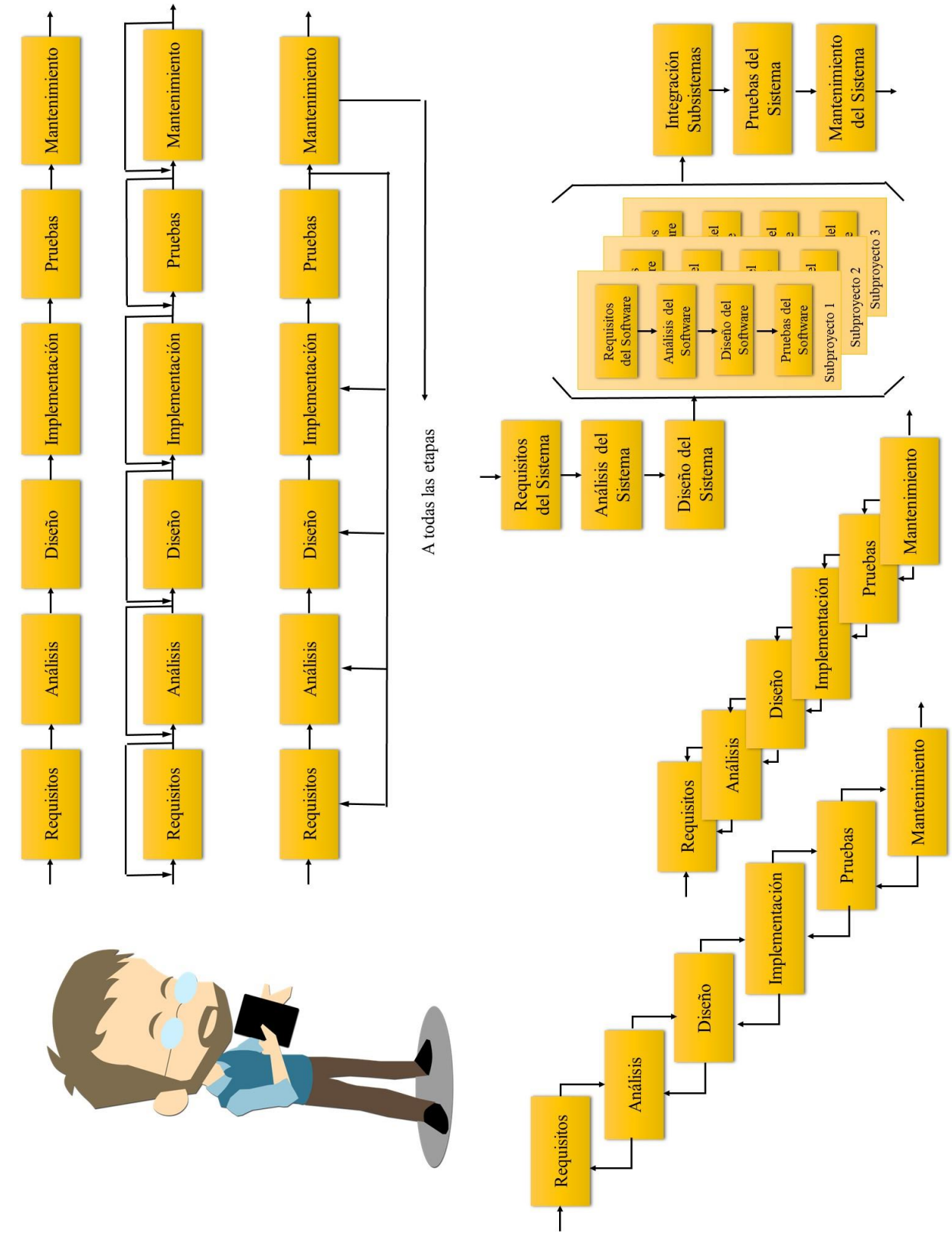


Figura 17: Ciclos de vida iterativo

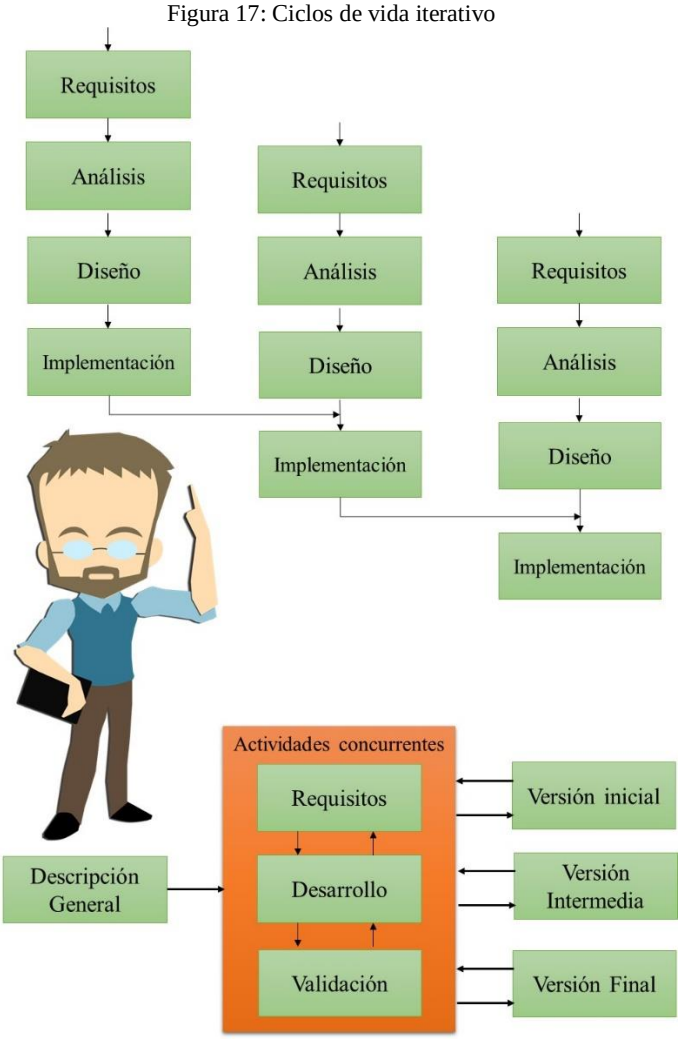
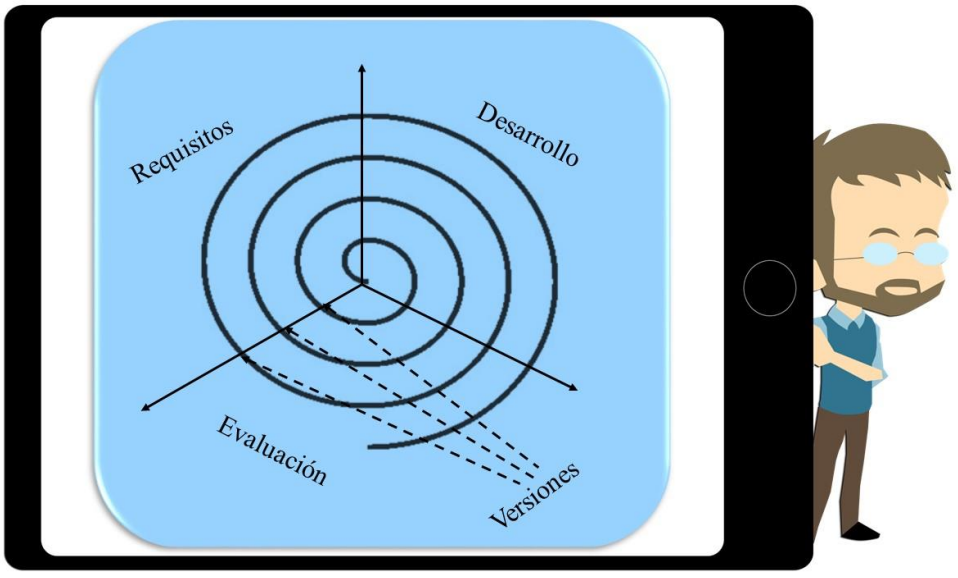


Figura 18: Ciclo de vida Espiral



Ciclo de vida ágil

El desarrollo ágil exige retroalimentación o *feedback* por parte de clientes-usuarios y de equipos multifuncionales para tener éxito. Es un enfoque en tiempo real a la gestión de proyectos que ayuda a gestionar el proceso de desarrollo a medida que se va construyendo, de forma progresiva a medida que va evolucionando. Se incorpora retroalimentación a pruebas y gestión al proyecto, todo ello de manera simultánea.

El ciclo ágil nace a principio de la década de los 90 en contraposición absoluta a lo que representaban las ciclos tradicionales (Leiva, 2013). A principios de la década de los 90, los ciclos tradicionales (cascada, iterativo, recursivo) constituían "el buen camino" para el desarrollo de software. Si bien es cierto, estos enfoques siguen siendo usados y son bastante efectivas cuando se trata de proyectos de gran envergadura, la rapidez con la que varía el entorno y la necesidad de hacer cambios con efectividad, han dado cabida a ciclos ágiles, orientados a elaborar un producto de calidad en un periodo de tiempo mucho más corto. Los primeros esfuerzos por agilizar el desarrollo de software fueron planteados por Martin (1990) y se dieron a conocer en la comunidad de Ingeniería de Software con el nombre de RAD (*Rapid Application Development*) el cual consistía en un entorno de desarrollo altamente productivo, en el que participaban equipos pequeños de personas utilizando software de desarrollo automático de código, tomando como entradas notaciones sintácticas de alto nivel. Estos esfuerzos desembocaron en la creación de una de las primeros enfoques consideradas oficialmente "Ágiles" en la historia de la Ingeniería de Software: *eXtreme Programming* (XP), fue el acrónimo que Kent Beck (1999) le dio a la programación en pareja, en donde la buena participación entre los miembros del equipo era un factor clave para que el proceso de desarrollo de software fuese rápido y eficaz.

Los procesos ágiles de desarrollo de software, también conocidos como metodologías livianas, intentan evitar los procesos clásicos de las metodologías tradicionales enfocándose en la gente y los resultados.

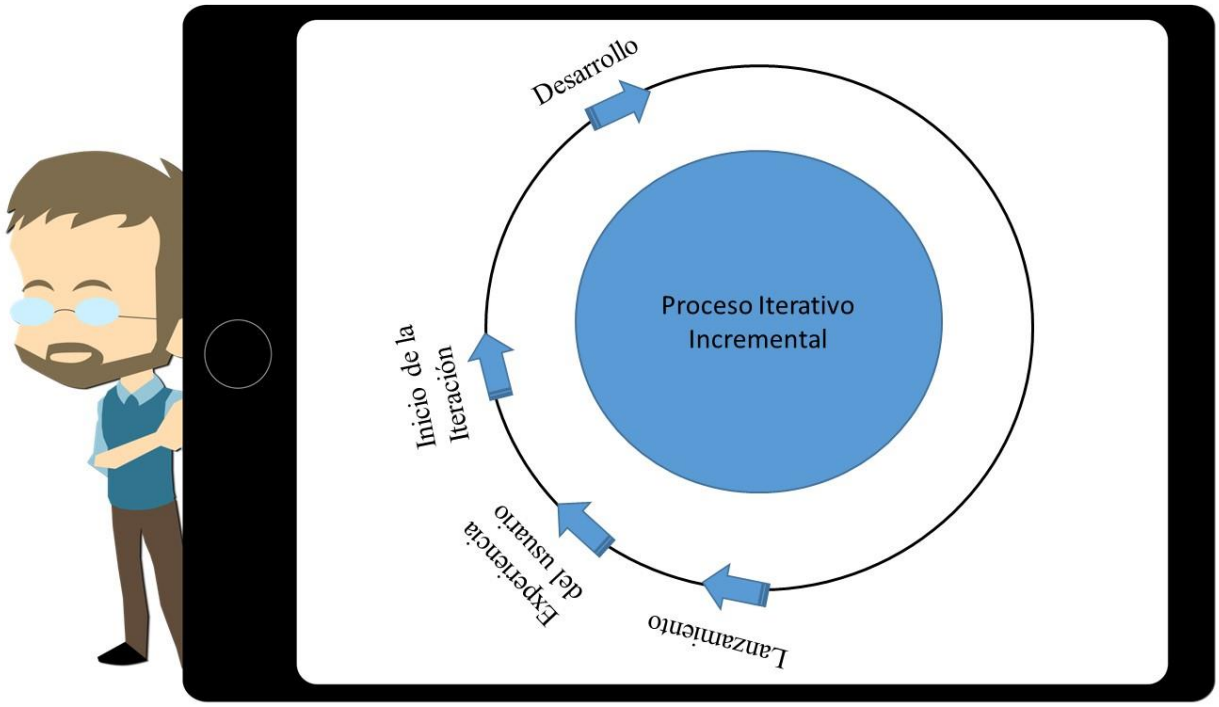
- Promueve iteraciones en el desarrollo.
- Su idea subyacente es generar software en plazos cortos de tiempos minimizando los riesgos.
- El software desarrollado en una unidad de tiempo es llamado una iteración (sprint), la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación.
- Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada iteración.
- Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.
- Los métodos Ágiles enfatizan las comunicaciones cara a cara a través de la documentación.
- La mayoría de los equipos Ágiles están localizados en una simple oficina abierta, a veces llamadas "plataformas de lanzamiento" (*bullpen* en inglés).
- La oficina debe incluir revisores, diseñadores de iteración, escritores de documentación y ayuda y directores de proyecto.
- Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso.

Características del desarrollo ágil:

- Proceso iterativo e incremental
- Mitigación del riesgo mediante iteraciones fijas
- Mejora continua
- Calidad desde el primer día
- Priorización de requerimientos de acuerdo a su valor
- Equipos dedicados y auto-gestionados
- Colaboración continua con el cliente
- Incorporar al cambio
- Prácticas de desarrollo modernas
- Declaración de principios y valores

En la figura 19, se muestra un ciclo de vida ágil genérico.

Figura 19: Ciclo de vida ágil



4.4 ETAPAS DE DESARROLLO DE SOFTWARE

En esta sección se abordarán las distintas etapas para el desarrollo de software. Es importante destacar que solo se tratarán las etapas de Requisitos, Diseño (IGU), Pruebas y Mantenimiento. El Análisis, Diseño (Arquitectura y Bases de Datos), Programación se ven en cursos anteriores de manera individual.

Gestión de requisitos

Definición de Requerimiento:

- "Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo". (Std 610.12-1900, IEEE: 62)
- "Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal". (Std 610.12-1900, IEEE:62)
- "Un requerimiento es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste". (Sommerville, 2005: 108)

Ejemplo:

Supóngase que hay que desarrollar el software para un sistema de control de una caldera de vapor. Posibles requisitos serían:

- El sistema evitará que el agua entre en ebullición
- El sistema leerá la temperatura del agua por medio del sensor
- El sistema podrá subir la temperatura del agua por medio del regulador

Clasificación:

- En funcionales vs. no funcionales (capacidades vs. restricciones)
- Por prioridades
- Por costo de implementación
- Por niveles (alto nivel, bajo nivel)
- Según su volatilidad/estabilidad
- Si son requisitos sobre el proceso o sobre el producto

No funcionales:

En la figura 20, se pueden apreciar los distintos requisitos de tipo no funcionales.

Fuentes de requisitos:

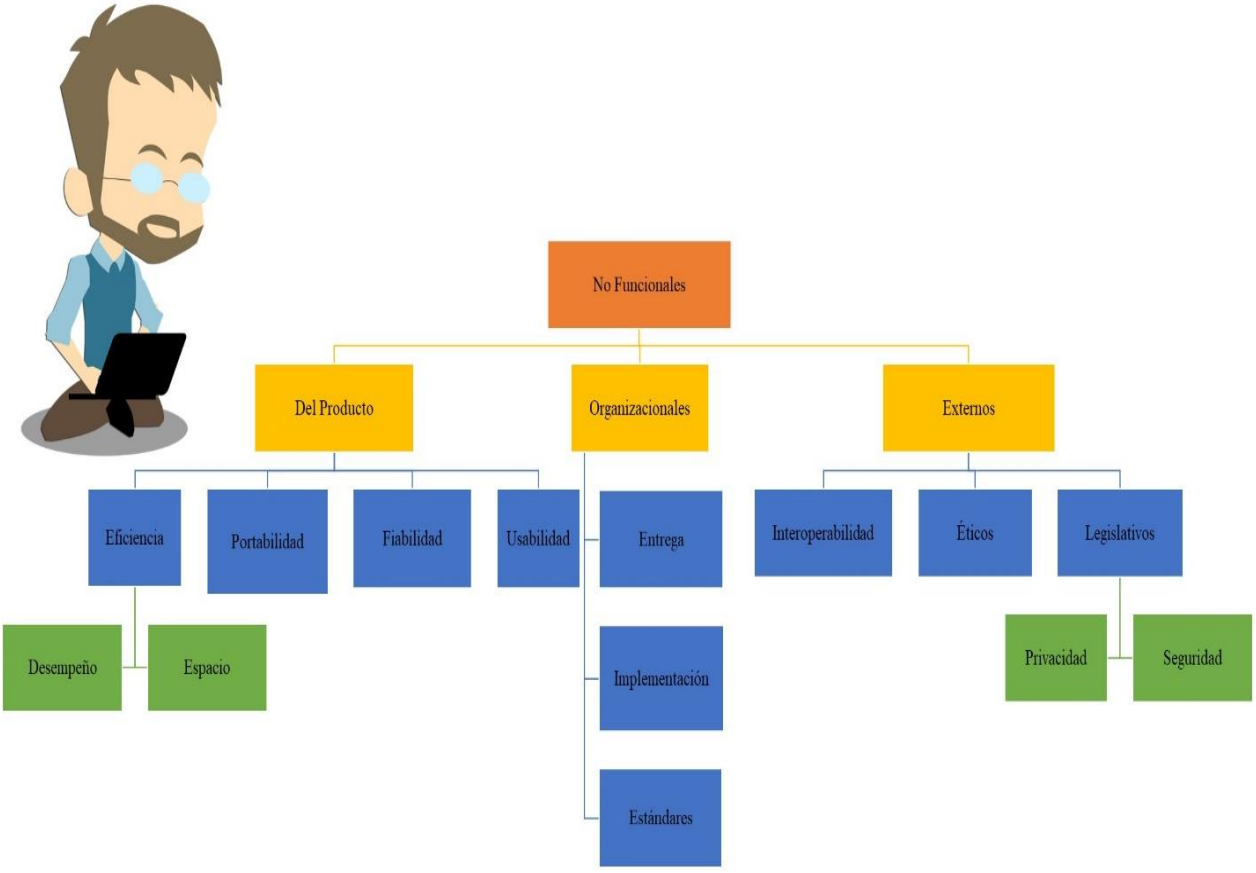
- Metas: Factores críticos de éxito.
- Conocimiento del dominio de la aplicación.
- Los interesados. Los afectados por el sistema.
- El entorno físico que rodea al sistema.
- El entorno organizacional.
- Los procesos de negocio.

Problemas de la Especificación:

- Los usuarios no pueden/saben describir muchas de sus tareas.

- Mucha información importante no llega a verbalizarse.
- A veces hay que “inventar” los requisitos (sistemas orientados a miles de usuarios).
- La Especificación no debería ser un proceso pasivo, sino cooperativo.

Figura 20: Requisitos no funcionales



En la tabla 24, se pueden apreciar las características deseables de una buena especificación de requisitos

Tabla 24: Características deseables de una buena especificación de requisitos

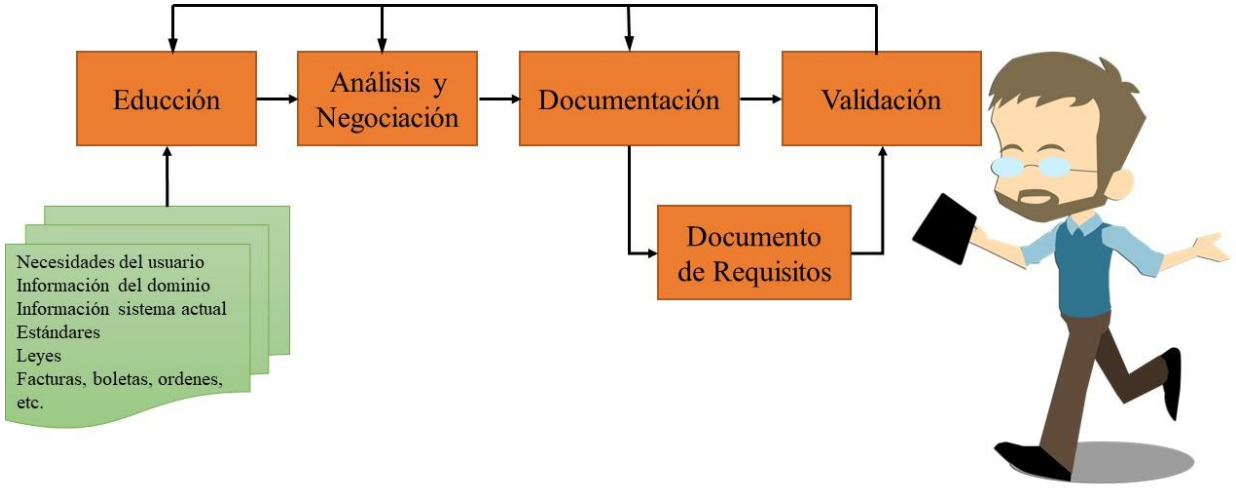
Característica	***Descripción***
No ambigua	Si todo requisito posee una sola interpretación
Completa	Si todo lo que se supone que el software debe hacer está incluido en la ERS. Por completitud, deberían describirse todas las posibles respuestas a todas las posibles entradas y en todas las situaciones. Además, la ERS no contendrá secciones del tipo “por determinar…”
Correcta	Todo requisito de la ERS contribuye a satisfacer una necesidad real (c/r al cliente)
Comprensible	Todo tipo de lectores (clientes, usuarios, desarrolladores, equipo de pruebas, gestores, etc.) entienden la ERS
Internamente consistente	No existen subconjuntos de requisitos contradictorios
Externamente consistente	Ninguno de los requisitos está en contradicción con lo expresado en documentos de nivel superior. Por ejemplo, en un sistema (hardware + software), lo requisitos del software no pueden contradecir los requisitos del sistema o estudio de factibilidad
Realizable	Si, dados los actuales recursos, la ERS es implementable
No redundante	Coda requisito se expresa en un solo lugar de la ERS. La redundancia, de todas formas, no es del todo mala si aumenta la legibilidad
A un nivel de	Ni demasiado detallada ni demasiado vaga

detalle	
Precisa	Si se hace uso de valores numéricos para precisar las características del sistema. La precisión es aplicable, ante todo, a los requisitos no funcionales. Por ejemplo, no es útil "El tiempo de respuesta será más bien rápido", sino "el tiempo de respuesta será menos de dos segundos". Esto en la medida de lo posible
Concisa	Debe ser lo más breve posible, sin que esto afecte al resto de atributos de calidad
Independiente del Diseño	Existen más de un diseño e implementación que realizan la ERS. Para ello la ERS debería limitarse a describir el comportamiento externo del sistema
Trazable	Cada requisito se puede referenciar de forma unívoca. Es fundamental para precisar qué requisitos son implementados por qué componente del diseño, lo cual es imprescindible a la hora de realizar pruebas de dicho componente
Modificable	Los cambios son fáciles de introducir
Electrónicamente almacenada	Se encuentra en un archivo de texto, en una base de datos, o mejor aún, ha sido creada con una herramienta de gestión de requisitos
Anotada por relativa importancia	Si los requisitos se clasifican según si importancia. Como mínimo un requisitos puede ser "obligatorios", "deseable", u "opcional". Esto sirve para no asignar demasiados recursos a la implementación de requisitos no esenciales
Trazada	Si está claro el origen de cada requisito (quién o qué lo pide)
Organizada	Si el lector puede fácilmente encontrar la información buscada
Con referencias cruzadas	Si se utilizan referencias entre requisitos relacionados (trazabilidad intra-ERS) o entre secciones relacionadas

El proceso de requisitos:

En la figura 21, se presentan las etapas del proceso de requisitos.

Figura 21: Proceso de Requisitos



Educción:

- Extracción es el nombre comúnmente dado a las actividades involucradas en el descubrimiento de los requerimientos del sistema.
- Aquí, los analistas de requerimientos deben trabajar junto al cliente para descubrir el problema que el sistema debe resolver, los diferentes servicios que el sistema debe prestar, las restricciones que se pueden presentar, etc.
- Demostración de que los requerimientos que definen el sistema son lo que el cliente realmente quiere.
- Los costos de errores en los requerimientos son altos, por lo cual, la validación es muy

importante. Fijar un error de requerimiento después del desarrollo puede resultar en un costo 100 veces mayor que fijar un error en la implementación.

El procedimiento para realizar una entrevista, puede ser visto en el anexo procedimientos del presente documento.

Análisis:

- Fase en la cual se enfoca en descubrir problemas con los requerimientos del sistema identificados hasta el momento.
- Usualmente se hace un análisis luego de haber producido un bosquejo inicial del documento de requerimientos.
- En esta etapa se leen los requerimientos, se conceptúan, se investigan, se intercambian ideas con el resto del equipo, se resaltan los problemas, se buscan alternativas y soluciones, y luego se van fijando reuniones con el cliente para discutir los requerimientos.
- No es posible convertir el análisis en un proceso estructurado y sistemático, lo que convierte a esta etapa en "subjetiva" porque depende en gran medida del juicio y de la experiencia.

Documentación:

- Se documentan los requerimientos acordados con el cliente, en un nivel apropiado de detalle.
- En la práctica, esta etapa se va realizando conjuntamente con el análisis, se puede decir que la especificación es el "pasar en limpio" el análisis realizado.
- Se utilizan técnicas y estándares, como el lenguaje unificado de modelado (UML).
- Se utiliza cada vez más la obtención de requerimientos basada en casos de uso.

Validación:

- Su objetivo es ratificar los requerimientos. Requerimientos consistentes y completos.
- La validación representa un punto de control interno y externo: Interno, porque se debe verificar internamente lo que se está haciendo; Externo, porque se debe validar con el cliente.
- Preferentemente, el documento de requerimientos obtenido en la etapa anterior sólo debería incluir los requerimientos que son aceptables para los usuarios.
- En definitiva, la validación de especificaciones realmente significa asegurarse de que el documento de requerimientos represente una descripción clara del sistema, y es una verificación final de que los requerimientos cubren las necesidades de los usuarios.
- Esta etapa puede confundirse con la de análisis. Mientras que en el análisis se trabaja sobre el boceto del documento de requerimientos, en la validación se utiliza el documento final.

Interfaz Gráfica de Usuario (IGU)

La interfaz es una parte muy importante del éxito o fracaso de una aplicación. La interfaz es del 47% al 60% de las líneas de código (McIntyre, 90). Un 48% del código de la aplicación está dedicado al desarrollo de la interfaz (Myers, 92) (véase Abascal, et al., 2001).

Conceptos:

Usuario: Persona que interacciona con un sistema informático. Persona -Individuo de la especie humana. Interacción -Todos los intercambios que suceden entre la persona y el computador (Baecker and Buxton, 1987).

Interfaz: Lenguaje de entrada para el usuario, un lenguaje de salida para el computador y un protocolo para la interacción. Los aspectos del sistema con los que el usuario entra en contacto.

Superficie de contacto: Una interfaz es una superficie de contacto y refleja las propiedades físicas de los que interactúan, se tienen que intuir las funciones a realizar y nos da un balance de poder y control. Donde los bits y las personas se encuentran (Negroponte, 1994). Es todo lo que el usuario experimenta, ve y hace con el sistema informático.

Accesibilidad: Asegurar que las personas son capaces de utilizar el producto. Acceso para todo el mundo.

Usabilidad: La medida (métrica) en la que un producto se puede usar por determinados usuarios para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado.

- La IGU es la puerta del usuario a la funcionalidad del sistema subyacente.
- IGU mal diseñadas es un factor que frena el uso de las funcionalidades.
- Es por tanto muy importante diseñar IGU usables.

Para poder hacer sistemas usables hace falta:

- Comprender los factores (psicológicos, ergonómicos, organizativos y sociales)
- Desarrollar herramientas y técnicas
- Conseguir una interacción eficiente, efectiva y segura

Características de la usabilidad:

- Fácil de aprender: Permite a los nuevos usuarios comprender como utilizar un sistema interactivo y llegar a un nivel máximo de conocimiento y uso.

  - ✓ Predictivo: Los conocimientos adquiridos en la historia de interacciones sean suficientes para determinar los resultados de la interacción futura.
  - ✓ Sintetizable: Nos permite conocer el estado de las interacciones pasadas.
  - ✓ Familiar: Si hay correlación entre el conocimiento que tiene el usuario y el conocimiento que se requiere para una interacción efectiva.
  - ✓ Consistencia: Es una característica fundamental. Para que un sistema interactivo sea consistente todos los mecanismos tienen que ser usados de la misma manera, siempre que se utilicen y sea cuando sea que se utilicen. Consejos para hacer una aplicación consistente:

- Seguir guías de estilo siempre que sea posible
- No cambiar alguna cosa si no se necesita cambiar
- Añadir nuevas funcionalidades en el conjunto preexistente
- No pedir al usuario cambiar las que ya conoce

Flexible: Un sistema interactivo es flexible cuando el usuario y el sistema pueden intercambiar información de muchas maneras.

Sólido: Un sistema interactivo es sólido cuando nos permite realizar los objetivos sin problemas.

- Recuperabilidad. El sistema es capaz de recuperarse de un error.
- Tiempo de respuesta. Tiempo que necesita el sistema para expresar los cambios de estado al usuario. Ha de ser un tiempo razonable.
- Adecuación a las tareas. El sistema permite todas las tareas que el usuario quiere hacer.

Modelo mental - metáfora

Modelo mental

Supuestos profundamente arraigados, generalizaciones e imágenes que influyen sobre el propio modo de comprender el mundo y actuar en él. En general, no se tiene conciencia de ellos o de los efectos que tienen sobre la conducta.

- Definen como se percibe, se siente, se piensa y se actúa.
- Son el contexto desde el cual se “escucha”.
- Los modelos mentales son los anteojos a través de los cuales se ve el mundo y se interpreta.
- Son fundamentales para la existencia humana.
- Son inconscientes, en general.
- Son diferentes de una persona a otra.

Fuentes de los Modelos Mentales

- La Biología
- El lenguaje (capacidad exclusiva del ser humano; especializado, fisiológico y psíquico. Nos capacita para abstraer, conceptualizar y comunicar)
- La cultura
- La Experiencia personal

Modelo mental – IGU

Un buen modelo mental permite a los usuarios predecir el efecto de sus acciones entender la relación entre el control de un dispositivo y el resultado exitoso. Un modelo mental pobre obliga al usuario a operar mecánicamente, confundido. Hace difícil determinar el efecto de la acción. Hace difícil de imaginar qué hacer en una situación nueva.

En la interacción se hace uso de la información adquirida por nuestros procesos perceptuales y que está almacenada en la memoria. Está organizada en estructuras semánticas. Los modelos mentales son las estructuras más relevantes en IGU.

¿Qué es el Modelo mental?

Durante el aprendizaje, una persona adquiere conocimientos de las relaciones estructurales y el funcionamiento del sistema con el que está interactuando. Este conocimiento se denomina modelo mental.

Modelo conceptual del sistema que el usuario tiene y que incluye la representación de su estructura y su funcionamiento (Norman 1983). No implica saber cómo funciona el sistema internamente. En general tiene un conocimiento mínimo del funcionamiento interno, es más bien una analogía. Modelo conceptual. Memoria largo plazo. Contiene información general y duradera.

Modelo mental. Memoria de trabajo. Representación más dinámica que sea capaz de adaptar la información almacenada en la memoria a largo plazo a las características específicas de la tarea que esté realizando la persona. La representación:

- Es incompleta
- Es ejecutable mentalmente, el usuario puede mentalmente simular su funcionamiento
- Es inestable, el usuario olvida los detalles
- No tiene unos límites claros, se confunde con los modelos mentales de sistemas físicos similares
- Es acientífica e incluye supersticiones y creencias erróneas sobre la conducta del sistema
- Es ahorrativa porque los usuarios prefieren reducir su complejidad

Modelo mental del computador

- Los computadores son mucho más complejos que los objetos físicos
- Como una máquina de propósito general, un computador no tiene un único modelo mental
- Los computadores son usados para varios propósitos y ejecutan diferentes aplicaciones. No es posible un único modelo mental
- Los diseñadores deben presentar un buen modelo subyacente que ayude a los usuarios a entender a cómo usar el sistema.
- Los diseñadores usan METÁFORAS para ayudar al usuario a adquirir un modelo mental adecuado.

¿Qué es una metáfora?

Las Metáforas son una parte integral de nuestro lenguaje. Se encuentran metáforas en:

- Las novelas, en la literatura y poesía, para proporcionar una imagen más colorida a los lectores.
- En el lenguaje de cada día y en las tareas diarias.

Las metáforas están omnipresentes. Muchas veces no se nota que se están usando. Permite

comunicar un concepto abstracto de una manera más familiar y accesible. Se utilizan continuamente:

- Los argumentos pueden ser defendidos o atacados, si una posición es indefendible se puede retirarse.
- Ahorrar, gastar, desaprovechar el tiempo.

Uso de las metáforas

- Normalmente los diseñadores emplean metáforas para ayudar al usuario a tener un modelo mental adecuado.
- Las metáforas pueden ser usadas para desarrollar interfaces adecuadas para las aplicaciones.
- La metáfora también puede ser aplicada a través de analogías, aunque la metáfora no esté en forma concreta en la interfaz de usuario (ejemplo, uso de un procesador de palabra en forma similar a una máquina de escribir).

En la tabla 25, se puede apreciar el uso de las metáforas.

Tabla 25: Uso de las metáforas (Abascal, et al., 2001)

Área de aplicación	***Metáfora***	***Conocimiento familiar***
Sistemas operativos	Escritorio	Tareas de una oficina
Entorno Orientado-objetos	Mundo físico	Comportamiento del mundo real
Hipertexto	Tarjetas de notas	Organización flexible de texto estructurado
Entorno de aprendizaje	Viaje	Tours, guías, navegación
Almacenamiento de archivos	Pila	Categorización de objetos en términos de la urgencia, proyectos, etc.
Entorno multimedios	Habitación (cada uno asociado con un medio/tarea)	Estructura espacial de un edificio
Trabajo colaborativo soportado por computador	Multi-agentes	Agente de viajes, mayordomos, y otros roles de servicios

Metáfora visual

Xerox, primera metáfora visual. Metáfora de la interfaz basada en la oficina física. La metáfora de organización global que se presentaba en la pantalla fue la de la sobremesa y se parecía al área de trabajo de una típica mesa de oficina.

- La metáfora visual forma parte de la interfaz.
- Las metáforas visuales combinan el sistema y los dominios familiares en una sola entidad. Los usuarios desarrollan modelos mentales más similares a la metáfora que al mundo real. La metáfora es el modelo que se aprende.
- Las metáforas se basan en asociaciones percibidas de manera similar por el diseñador y el usuario. Si el usuario no tiene la misma base cultural que el desarrollador es fácil que la metáfora falle.

¿Qué es una metáfora visual?

Es una imagen que nos permite representar alguna cosa y que el usuario puede reconocer lo que representa y por extensión comprender su propósito.

Intuición

Las personas entendemos las metáforas por intuición. Cognición inmediata. Conocimiento de una cosa obtenida sin utilizar inferencia o razonamiento.

¿Cómo funcionan estas metáforas?

Las carpetas son contenedores de documentos en el mundo real y en el virtual. Abres una carpeta para tomar o dejar alguna cosa, podemos poner carpetas dentro de carpetas y dentro de carpetas, se puede mover las carpetas por toda la sobremesa. Algunas propiedades físicas están ausentes. No pesan no hacen ruido cuando se abren, pero por otra parte tienen propiedades mágicas, puedes poner el mismo documento en dos carpetas al mismo tiempo, aunque puedas pensar que los puedes pasar por una copiador virtual, puedes reproducir un conjunto de carpetas y sus documentos automáticamente, puedes ordenar las carpetas por orden alfabético.

Metáforas compuestas

La metáfora de la sobremesa se ha combinado con otras metáforas para permitir a los usuarios que puedan hacer un conjunto de tareas más amplio.

Ejemplos:

- La barra de desplazamiento como la metáfora del rollo
- Menús y ventanas
- Cortar y pegar que se basa en el diseño de páginas de diarios murales, por ejemplo (composición).

Objetivo de la metáfora

Se puede considerar que puede llegar a ser la metáfora transparente al usuario y no tener ningún esfuerzo cognitivo.

Diseño de metáforas- lenguaje visual

Las metáforas pueden conseguir su efectividad a través de la asociación con organizaciones. Se puede asociar estructuras, clases, objetos, atributos a nombres u operaciones. Se puede asociar procesos, algoritmos a verbos.

El procedimiento para diseñar un lenguaje visual (metáforas), puede ser visto en el anexo procedimientos del presente documento.

Evaluación de la metáfora visual

La evaluación consta de diversas metodologías y técnicas que estudian la usabilidad de un sistema interactivo en diferentes etapas del ciclo de vida. Aplicar los métodos de evaluación de la usabilidad permite crear mejores productos y ayudar a los usuarios a realizar sus tareas más productivamente.

Ventajas de la evaluación

- Una reducción de los costos de producción.
- Reducción de los costos de mantenimiento y apoyo.
- Reducción de los costos de uso.
- Mejora en la calidad del producto.

Inspección – heurística

Un conjunto reducido de evaluadores evalúan la interfaz teniendo en cuenta los principios reconocidos de usabilidad la heurística.

Se utilizan expertos para validar la interfaz, porque es difícil que un evaluador pueda encontrar todos los problemas de usabilidad en una interfaz a partir de unos criterios definidos.

Cada evaluador realiza individualmente una revisión de la interfaz. Al terminar las evaluaciones se les permite comunicar los resultados y sintetizarlos. Este procedimiento es importante para asegurar unas evaluaciones independientes e imparciales de cada de ellos. Los resultados de la evaluación se pueden registrar con informes escritos o haciendo que comuniquen verbalmente sus comentarios a un observador mientras realizan la evaluación.

10 reglas heurísticas:

- El estado del sistema debe de estar siempre visible
- Se debe de utilizar el lenguaje de los usuarios
- El usuario tiene control y libertad
- Hay consistencia y se siguen estándares
- Existe prevención de errores
- Se minimiza la carga de la memoria del usuario
- Existe flexibilidad y eficiencia de uso
- Los diálogos son estéticos y diseño minimalista
- Existe ayuda y documentación
- Al utilizar la ayuda, se reconocen, diagnostican, y se recuperan

Pruebas del Software

Hasta la fecha no se ha desarrollado ninguna teoría universalmente aceptada acerca de la prueba de software. Lo único que hay es un conjunto de aproximaciones metodológicas que facilitan y hacen más eficiente el proceso de prueba.

Se llama PRUEBA del Software al proceso en el que se ejecuta un sistema con el objetivo de detectar fallos.

Se llama DEPURACIÓN al proceso en el que se localiza el defecto que es la causa de un fallo, se determina la forma de corregirlo, se evalúa el efecto de la corrección y se lleva a cabo la corrección.

Por lo general, después del proceso de depuración será necesario repetir el proceso de prueba, para garantizar que el defecto quedó efectivamente corregido y que no se introdujeron nuevos defectos.

El costo de detección de los defectos suele ser mucho mayor que el costo de corrección de los mismos, y este es un punto en contra de las pruebas como técnica de control de calidad, ya que siempre es necesario un paso de diagnóstico hasta que se localiza la causa de los fallos.

En otras actividades de control de calidad (próximo capítulo), por el contrario, como pueden ser las revisiones, se localizan directamente los defectos, no sus síntomas, por lo que se ahorra el proceso de diagnóstico.

En un proyecto grande, la prueba se puede llevar hasta el 50 o 60% del esfuerzo dedicado al proyecto. Por eso es muy importante seleccionar bien las pruebas que se van a realizar, teniendo en cuenta que sólo las pruebas que revelan defectos son las que realmente merecen la pena.

El objetivo del proceso de prueba no es, como pudiera parecer, demostrar que el software está libre de defectos, sino precisamente descubrir defectos.

Proceso de pruebas

El proceso de prueba conlleva la realización de un conjunto de tareas a lo largo del ciclo de vida del sistema.

De acuerdo con el estándar IEEE 1012-1986 el conjunto mínimo de pruebas que se deben realizar es:

- Prueba modular, prueba unitaria o prueba de componentes
- Prueba de integración
- Prueba del sistema
- Prueba de aceptación
- Prueba de regresión

La prueba modular consiste en la prueba de cada módulo aislado del resto del sistema.

La prueba de integración se realiza a medida que los diferentes módulos del sistema se integran en el mismo. Ya se ha realizado la prueba modular, y se supone que todos módulos son correctos. El objetivo fundamental de esta prueba es comprobar que las interfaces entre los distintos módulos son correctas. Algunas de las comprobaciones que es necesario realizar son:

- Corrección en la sintaxis en la invocación de procedimientos y funciones.

- Compatibilidad de tipos entre los argumentos del procedimiento o función y los parámetros de llamada.
- Corrección y completitud de las especificaciones de los módulos.

Se pueden utilizar tres posibles estrategias de integración:

- De arriba a abajo (*top-down*): Consiste en empezar la integración y la prueba por los módulos que están en los niveles superiores de abstracción, e integrar incrementalmente los niveles inferiores.
- De abajo a arriba (*bottom-up*): Consiste en empezar la integración y la prueba por los módulos que están en los niveles inferiores de abstracción, e integrar incrementalmente los niveles superiores.
- De *big-bang*: Consiste en integrar y probar todo al mismo tiempo.

La prueba del sistema se realiza cuando se han integrado todos los módulos, y su objetivo es comprobar que el sistema satisface los requisitos del usuario, tanto los funcionales como los no funcionales.

La prueba de aceptación se realiza una vez que el sistema se ha implantado en su entorno real de funcionamiento, y su objetivo es demostrar al usuario que el sistema satisface sus necesidades.

La prueba de regresión tiene como objetivo comprobar que toda nueva versión de un producto software es de no menos calidad que la versión anterior, es decir, que al introducir cambios no se ha reducido la valoración de ninguna de las características de calidad que tenía el producto.

En la figura 22, se puede apreciar la relación entre las pruebas y los distintos subproductos del proceso de desarrollo, conocida como estrategia “V”.

Mantenimiento del Software

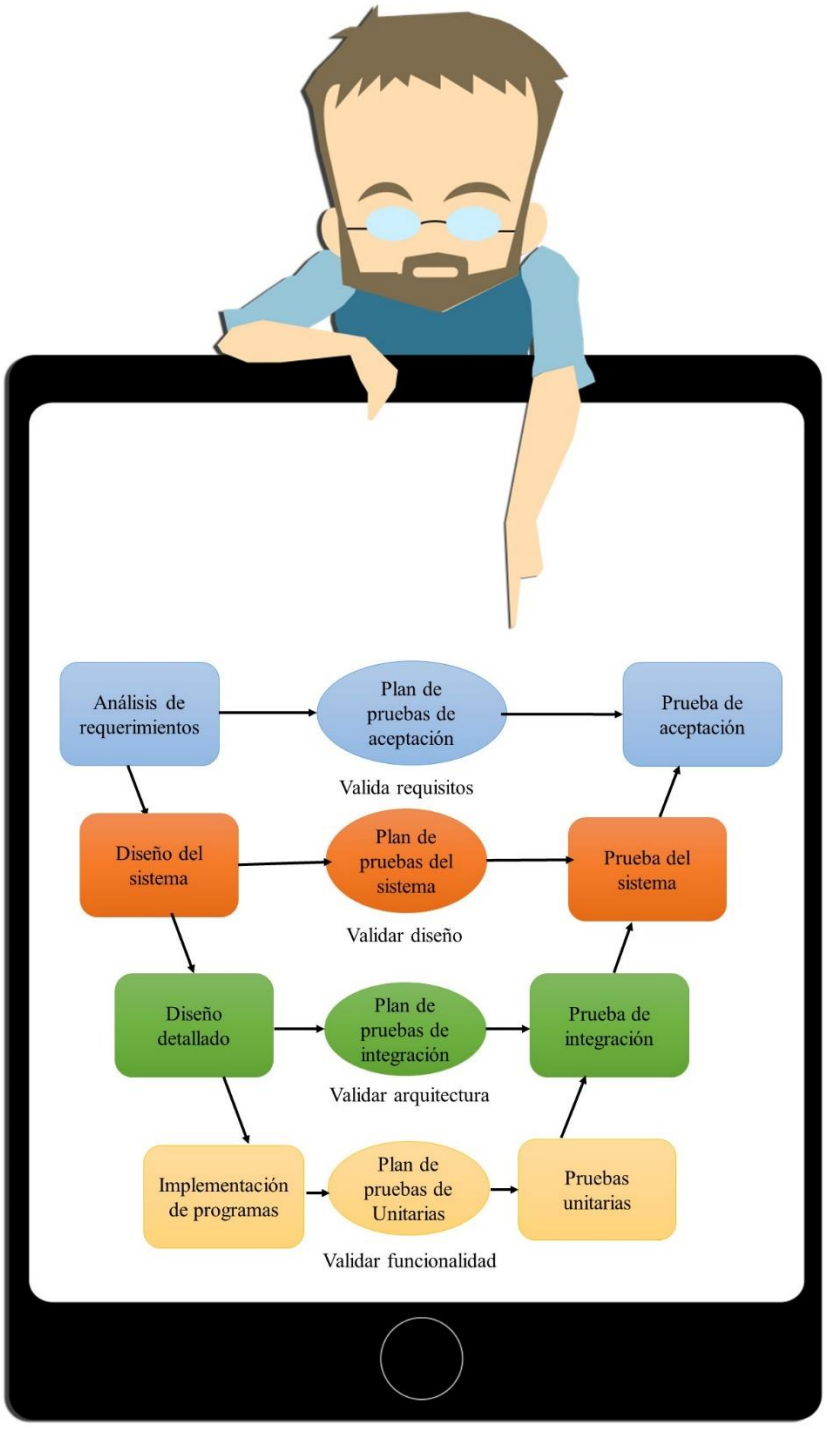
Definición

Mantenimiento: La modificación de un producto software después de su entrega al cliente o usuario para corregir defectos, para mejorar el rendimiento u otras propiedades deseables, o para adaptarlo a un cambio de entorno (Estándar IEEE 1219).

Mantenibilidad: Capacidad de un producto software para ser modificado. Estas modificaciones incluyen correcciones, mejoras, o adaptaciones a cambios en el entorno, los requisitos o las especificaciones funcionales (Estándar IEEE 1219).

Sistema Heredado: Es un método, tecnología, computador o aplicación antiguo que continúa en uso porque aún satisface las necesidades de los usuarios, aun existiendo nuevas tecnologías o métodos más eficientes disponibles (ej. COBOL en la banca o sistema financiero).

Figura 22: Proceso de pruebas “V”



Tipos de mantenimiento

Correctivo: Aunque se hayan tomado las mejores medidas y actividades de garantía de calidad, es muy probable que el cliente descubra defectos en el software. El mantenimiento correctivo cambia el software para corregir los errores (Ej. IVA cambia del 15% al 19%)

Adaptativo: Con el paso del tiempo es probable que cambie el entorno original (Sistema Operativo, periféricos, UCP) para el que se desarrolló el software. El mantenimiento adaptativo consiste en modificar el software para acomodarlo a los cambios de su entorno externo.

Perfectivo: Conforme utilice el software el cliente/usuario puede descubrir funciones adicionales que podrían ser incorporadas al software. El mantenimiento perfectivo amplía el software más allá de sus requisitos funcionales originales (Ej. Firma digital en banca *online*).

Preventivo: Modificación para detectar y corregir fatos latentes antes que se conviertan en fallos operacionales. Mejorar las propiedades del software (Ej. Recodificar para aplicar patrones de diseño).

Retos del mantenimiento

- Efecto iceberg
- No es lo mismo producir que mantener: 40 LDC desarrollada por 1 LDC mantenida
- Código Heredado

  - ✓ Desarrollado con tecnologías y técnicas "anticuadas"
  - ✓ No hay documentación
  - ✓ Si la hay, está en notación ad-hoc a los desarrolladores, que ya no trabajan en la empresa
  - ✓ Reescribirlo entero no es factible
  - ✓ El sistema no tiene por qué estar bien diseñado, programado, ni haber sido desarrollado siguiendo un proceso de ingeniería

Solución

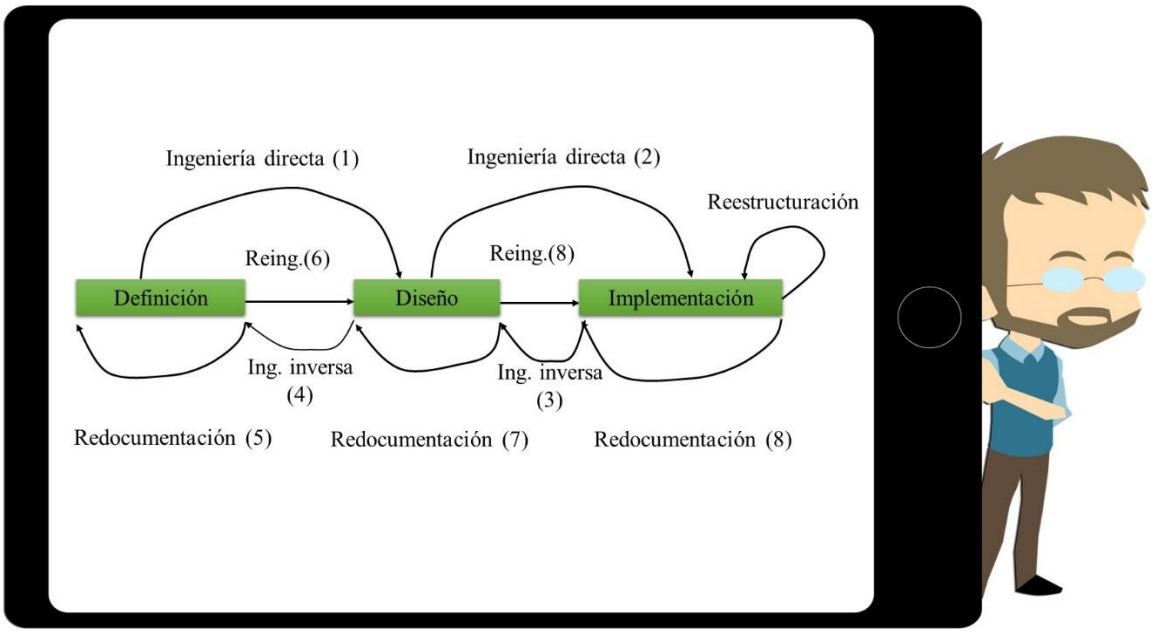
- Establecer procedimiento claramente definidos y estandarizados
- Asignarle recursos adecuados, tanto físicos y económicos como humanos
- Usar técnicas de control de calidad, tanto sobre el producto como sobre el proceso

Técnicas

- Reingeniería: examen y modificación del sistema para reconstruirlo en una nueva forma.
- Ingeniería inversa: análisis de un sistema para identificar sus componentes y las relaciones entre ellos, así como para crear representaciones del sistema en otra forma o en un nivel de abstracción más elevado.
- Reestructuración del software: consiste en la modificación del software para hacerlo más fácil de entender y cambiar o menos susceptible de incluir errores en cambios posteriores.
- Transformación de programas: técnica formal de transformación de programas.

En la figura 23, se puede apreciar la relación entre las distintas técnicas de mantenimiento de software.

Figura 23: relación entre las distintas técnicas de mantenimiento



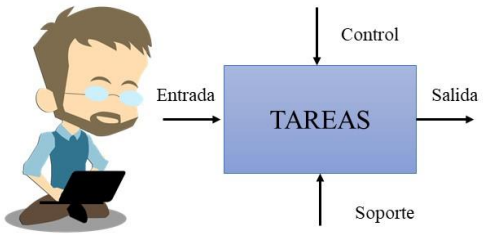
Proceso de mantenimiento

El proceso está de acuerdo al Std. IEEE 12207 – Std. ISO/IEC 14764, cuyas fases son:

- Implementación del proceso.
- Análisis de problemas y modificaciones.
- Implementación de las modificaciones.
- Revisión y aceptación del mantenimiento.
- Migración.
- Retiro del software.

Cada una de estas actividades se compone de tareas, y cada tarea describe una acción específica con las entradas y salidas. Una tarea especifica qué hacer, pero no cómo hacerlo. En la figura 24 se muestra la estructura de las actividades.

Figura 24: Estructura de las actividades



Donde:

- Las entradas se refieren a los elementos que se utilizan en la actividad de mantenimiento para

generar las salidas.
- Se necesitan controles efectivos para proporcionar una guía útil para que la actividad de mantenimiento produzca las salidas deseadas.
- Las salidas son los objetos generados por la actividad de mantenimiento.
- El soporte se refiere a los elementos que apoyan la actividad de mantenimiento.

Las tablas indicadas como [Tabla N°] se pueden encontrar en Herrera, 2015.

Implementación del proceso

Esta actividad establece planes y procedimientos que deben seguirse. Un plan de mantenimiento se realiza simultáneamente con el plan de desarrollo. Esta actividad posee 3 tareas:

1. Desarrollar un Plan de mantenimiento: El plan de mantenimiento describe una estrategia para mantener el sistema, mientras que los procedimientos de mantenimiento describen en detalle cómo lograr realmente el mantenimiento. El plan también se describe cómo:

   i. organizar y personal del equipo de mantenimiento
   ii. asignar responsabilidades entre los miembros del equipo
   iii. los recursos de programación. La idea principal es proporcionar apoyo económico para el equipo de mantenimiento

2. Gestionar las solicitudes de modificación: Los usuarios envían una solicitud de modificación (o modificar) y solicita comunicarse con el mantenedor. El mantenedor establece procedimientos para recibir, registrar y peticiones pista usuario para modificaciones y darles retroalimentación. El proceso de resolución del problema se inicia cuando se recibe un MR (*Modification request*). MR se clasifican por el mantenedor ya sea como informes de problemas (correctivo) o la mejora peticiones (adaptativo y perfectivo). El proceso de mantenimiento prioriza y seguimiento de estas solicitudes individualmente como diferentes tipos de mantenimiento.

3. Gestionar la configuración CM (*Configuration management*): El producto de software y los cambios realizados en él durante su vida útil de mantenimiento necesitan ser controlados. Básicamente, el control de cambio se lleva a cabo mediante la aplicación e implementación de un proceso de SCM (*software configuration management*) aprobado. El proceso SCM se implementa mediante el desarrollo y seguimiento de un plan de gestión de configuración CMP (*configuration management plan*) y los procedimientos correspondientes.

Análisis de problemas y modificaciones

Esta actividad se invoca después de las transiciones del sistema de software de la etapa de desarrollo a la etapa de mantenimiento y se llama iterativa cuando la necesidad de la modificación se plantea. Esta actividad posee 5 tareas:

1. Analizar MR: El mantenedor analiza el MR para determinar el impacto en la organización, el hardware, el sistema existente, otros sistemas de interconexión, la documentación, las estructuras de datos, y los seres humanos (operadores, mantenedores y usuarios). El objetivo

general de análisis de impacto es determinar todas las entidades que van a ser modificados y/o afectadas si el MR se va a implementar. Los pasos del análisis de impacto se dan en la [Tabla 3].

2. Verificar: El mantenedor deberá reproducir el problema y documentar los resultados de las pruebas en el entorno de laboratorio si el MR es correctivo con el fin de determinar la validez de la MR. Para las tareas de mantenimiento adaptativo y perfectivo, no se requiere verificación. El mantenedor diseña una estrategia de prueba para verificar y replicar el problema.

3. Proponer opciones: El mantenedor debe describir dos o más soluciones alternativas a la MR en base a los análisis realizados. El informe de soluciones alternativas debe incluir el costo, el esfuerzo, y el calendario para la aplicación de diferentes soluciones. El mantenedor deberá realizar los pasos de la tarea que se muestran en la [Tabla 4] para identificar soluciones alternativas al MR.

4. Documentar: El mantenedor documenta los MRs, los resultados del análisis y el informe opción de ejecución después de terminado el análisis y las soluciones alternativas se identifican. El mantenedor puede utilizar los pasos de la tarea que se muestran en la [Tabla 5] para escribir este documento.

5. Aprobar: El mantenedor presenta el informe de análisis a la autoridad competente en la organización para buscar su aprobación para la opción de cambio seleccionado. Una vez aprobado, el mantenedor actualiza los requisitos si el MR es una mejora.

Implementación de las modificaciones

En esta actividad, los mantenedores deben identificar los elementos para ser modificados y ejecutar un proceso de desarrollo a aplicar en la práctica las modificaciones.

En esta actividad, los mantenedores deben: (i) identificar los elementos para ser modificados y (ii) ejecutar un proceso de desarrollo a aplicar en la práctica las modificaciones.

El mantenedor determina el tipo de documentación, unidades de software, y la versión del software que han de ser cambiado. Aunque el desarrollo se convierte en parte de la actividad de modificación, que se adapta a eliminar las actividades que no se aplican al esfuerzo de mantenimiento, como la obtención de requisitos y diseño arquitectónico.

Para asegurar que lo modificado o los requisitos recién añadidos se implementan correctamente, los planes de prueba y los procedimientos están incluidos en el proceso de desarrollo. Además, se asegura que los requisitos que no han sido modificados no se ven afectados por la nueva implementación.

Las entradas a esta actividad incluyen todo el trabajo de análisis realizado en las actividades anteriores, y la salida es una nueva línea de base de software.

Revisión y aceptación del mantenimiento

Por medio de esta actividad, se garantiza: que los cambios realizados en el software son correctos y que se realizan cambios en el software de acuerdo con las normas y metodologías aceptadas.

Por medio de esta actividad, se garantiza: (i) los cambios realizados en el software son correctos y (ii) se realizan cambios en el software de acuerdo con las normas y metodologías aceptadas.

Además, se realizan los siguientes procesos:

i. un proceso para la gestión de la calidad
ii. un proceso para verificar el producto
iii. un proceso para validar el producto
iv. un proceso de revisión del producto

En el plan de mantenimiento debería haber sido documentado cómo se adaptaron estos procesos de apoyo para hacer frente a las características del producto de software específico. Las entradas a esta actividad incluyen el software modificado y los resultados de las pruebas.

La actividad de implementación del proceso consta de dos tareas principales: la revisión y aprobación. Los pasos de la tarea, tanto para su revisión y aprobación se enumeran en la [Tabla 6].

Migración

Esto se refiere al proceso de mover un sistema de software de un entorno tecnológico a una diferente que se considera que es mejor. Esta actividad posee 7 tareas:

1. Norma de migración: Durante la migración de un producto de software de un antiguo a un nuevo entorno operativo, el mantenedor debe garantizar que cualquier producto de software o datos adicionales producidos o modificados se adhieran a la norma ISO/IEC 12207. Como parte de las tareas estándar, el mantenedor (i) identifica todos los elementos de software o datos que han sido modificados o añadidos y (ii) garantiza que las tareas se realizaron de acuerdo con la norma ISO/IEC 12207.

2. Plan de migración: Para la migración exitosa, un plan debe ser desarrollado, documentado, revisado y ejecutado. El mantenedor realiza los pasos de la tarea que se muestran en la [Tabla 7] para escribir este documento. El plan se desarrolla en colaboración con los clientes y se ocupa de lo siguiente:

   - Análisis de requerimientos y definición de la migración.
   - Desarrollo de herramientas de migración.
   - La conversión de producto de software y datos.

3. Notificación de intención: El mantenedor explica a los usuarios de: (i) por qué el soporte para el entorno antiguo ha sido descontinuado; (ii) el nuevo entorno y cuándo va a ser objeto de soporte; y (iii) la disponibilidad de otras opciones, si hay alguna, tras la eliminación del entorno antiguo.

4. Implementación de operaciones y entrenamiento: Una vez que un producto de software se ha

mejorado mediante la modificación y probado por el mantenedor, se instala en un entorno operativo para funcionar simultáneamente con el sistema antiguo. Mediante la ejecución del antiguo y el nuevo sistema en paralelo, los usuarios tienen la oportunidad de familiarizarse con el nuevo sistema, por lo que la transición del antiguo al nuevo sistema se vuelve más paulatino. Además, se creará un entorno para el mantenedor a fin de comparar y entender las relaciones entrada/salida del antiguo y nuevo sistema. Los pasos que se indican en la [Tabla 8] se pueden realizar por el mantenedor en este paso.

5. La notificación de finalización: El mantenedor notifica a todos los sitios que el nuevo sistema entrará en funcionamiento y que el antiguo sistema se descontinua y será desinstalado, después de la finalización de la formación y el funcionamiento en paralelo de ambos sistemas. En esencia, los siguientes pasos de la tarea son realizadas por el mantenedor:

   i. Anunciar la migración.
   ii. Documento de los aspectos específicos del centro y hacer un plan para resolverlos.
   iii. Archivar el antiguo sistema, incluidos los datos y el software.
   iv. Retirar el equipo antiguo. Verificación de la migración.
   v. Compatibilidad con versiones anteriores con el entorno de ejecución antiguo.

6. Revisión después de la operación: Después de la instalación y operación de un sistema modificado, se realiza una revisión para evaluar el impacto de cambiar el sistema en el nuevo entorno. Los informes de revisión se envían a las partes competentes para la información, orientación, y demás acciones. El mantenedor ejecuta los siguientes pasos, como parte de la tarea:

   i. Analizar los resultados de la ejecución de los dos sistemas simultáneamente.
   ii. Identificar las áreas potenciales de riesgo.
   iii. Resumir las lecciones aprendidas.
   iv. Elaborar un informe sobre el análisis de impacto.

7. Archivos de datos: Los datos asociados con el antiguo entorno se hacen accesibles para cumplir con los requisitos contractuales de protección de datos y auditoría. El mantenedor realiza los siguientes pasos como parte de la tarea:

   i. Los datos y software antiguo se archivan.
   ii. Los datos y el software antiguo se ponen en múltiples respaldos.
   iii. Los respaldos se guardan en lugares seguros.

Retiro del software

Un producto de software se retira cuando se ve que ha llegado al final de su vida útil. Un análisis económico se realiza para retirar el producto y se incluye en el plan de retiro. A veces, el trabajo realizado por el producto ya no es necesario; por lo tanto, no se reemplaza el producto retirado. En otros casos, un nuevo producto de software ya se ha desarrollado para reemplazar el sistema actual. En cualquier caso, el sistema de software debe ser retirado del servicio de una manera ordenada. Además, las consideraciones son dadas para acceder a los datos producidos por el software para ser retirado. Todos los objetos de la actividad de retiro se controlan con CM. Esta actividad posee

5 tareas:

1. Plan de retiro: A fin de garantizar un retiro exitoso, un plan de retiro es desarrollado, documentado, revisado, y ejecutado. El mantenedor realiza los pasos de la tarea que se muestran en la [Tabla 9] para escribir este documento. El plan se desarrolla en colaboración con los clientes para hacer frente a lo siguiente:

   - La transición a un nuevo sistema de software.
   - Retiro de soporte parcial o total después de un período de gracia.
   - La responsabilidad de ningún soporte contractual futuro.
   - Archivar el sistema de software, incluyendo toda la documentación.
   - La accesibilidad a los datos archivados.

2. Notificación de intención: El mantenedor transmite a los usuarios: (i) la razón para discontinuar el apoyo para el producto; (ii) una nota sobre el reemplazo o actualización para el sistema a eliminar, con una fecha de disponibilidad; y (iii) una lista de las otras opciones disponibles, si hay alguna, tras la eliminación del entorno antiguo.

3. Implementar operaciones paralelas y formación: Si hay un sistema de reemplazo para el producto de software que se retiró, se instala en un entorno operativo para funcionar simultáneamente con el sistema antiguo. Mediante la ejecución de lo nuevo y lo del antiguo sistema en paralelo, los usuarios tendrán la oportunidad de familiarizarse con el nuevo sistema de modo que la transición de antiguo al nuevo sistema se vuelve más suave. Además, se creará un entorno para el mantenedor para comparar y entender las relaciones de entrada/ salida entre el nuevo sistema y el sistema antiguo. Además, se proporciona formación a los usuarios durante este período.

4. Notificación de finalización: El mantenedor notifica a todos los interesados que el nuevo sistema entrará en funcionamiento y que el antiguo sistema puede cerrarse. El antiguo sistema se suele cerrar después de que el nuevo sistema está en funcionamiento durante un cierto período de tiempo. El mantenedor realiza los pasos siguientes como parte de la tarea:

   i. Hacer un anuncio sobre los cambios.
   ii. Identificar los problemas específicos a los sitios individuales y describir cómo los va a resolver.
   iii. Almacenar los antiguos datos y software en un archivo.
   iv. Desconectar y mover la antigua infraestructura de hardware.

5. Archivos de datos: Los datos asociados o utilizados con el entorno anterior se pondrá a disposición de acuerdo con los requisitos contractuales relacionadas con la protección de datos y auditoría. El mantenedor ejecuta los siguientes pasos como parte de esta tarea:

   i. Archiva datos y software antiguos.
   ii. Realiza varias copias de datos y software antiguos.
   iii. Mantenga los respaldos en lugares seguros.

4.5 MÉTODOS PARA DESARROLLAR SOFTWARE

Método

Es un conjunto integrado de modelos, técnicas y herramientas, los que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo.

Se basan en una combinación de los modelos de proceso genéricos (cascada, incremental…). Definen artefactos, roles y actividades, junto con prácticas y técnicas recomendadas.

Para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto, para llevarlo a cabo con altas posibilidades de éxito.

Estrategia

Define una táctica global para enfrentarse con el proyecto. Entre los elementos que forman parte de una metodología se pueden destacar:

- Fases: tareas a realizar en cada fase.
- Productos: E/S de cada fase, documentos.
- Procedimientos y herramientas: apoyo a la realización de cada tarea.
- Criterios de evaluación: del proceso y del producto. Saber si se han logrado los objetivos.

Marco de Trabajo (*framework*)

Una gran variedad de estos marcos de trabajo han evolucionado durante los años, cada uno con sus propias fortalezas y debilidades. Una metodología de desarrollo de sistemas no tiene que ser necesariamente adecuada para usarla en todos los proyectos. Cada una de las metodologías disponibles es más adecuada para tipos específicos de proyectos, basados en consideraciones técnicas, organizacionales, de proyecto y de equipo.

Un método de desarrollo de software o método de desarrollo de sistemas en ingeniería de software es un marco de trabajo que se usa para estructurar, planificar y controlar el proceso de desarrollo de un software.

El marco de trabajo de un método de desarrollo de software consiste en:

- Una filosofía de desarrollo de software, con el enfoque o enfoques del proceso de desarrollo de software.
- Múltiples modelos, técnicas y herramientas para ayudar en el proceso de desarrollo de software.

Estos marcos de trabajo están con frecuencia vinculados a algunos tipos de organizaciones, que se encargan del desarrollo, soporte de uso y promoción de la metodología. Los métodos con frecuencia se documentan de alguna manera formal.

Ventajas de usarlo

Desde el punto de vista de gestión:

- Facilitar la tarea de planificación
- Facilitar la tarea del control y seguimiento de un proyecto
- Mejorar la relación costo/beneficio
- Optimizar el uso de recursos disponibles
- Facilitar la evaluación de resultados y cumplimiento de los objetivos
- Facilitar la comunicación efectiva entre usuarios y desarrolladores

Desde el punto de vista de los ingenieros del software:

- Ayudar a la comprensión del problema
- Optimizar el conjunto y cada una de las fases del proceso de desarrollo
- Facilitar el mantenimiento del producto final
- Permitir la reutilización de partes del producto

Desde el punto de vista del cliente o usuario:

- Garantía de un determinado nivel de calidad en el producto final
- Confianza en los plazos de tiempo fijados en la definición del proyecto
- Definir el ciclo de vida que más se adecue a las condiciones y características del desarrollo

Ejemplos

Clásicos:
- Propuesta de JGM
- RUP
- Prototipado - Arnowitz et al.

Ágiles:
- Scrum
- Feature Driven Development (FDD)

Prototipado de Software

¿Qué es un prototipo?:

Un prototipo en sentido genérico es una implementación parcial pero concreta de un sistema o una parte del mismo que principalmente se crean para explorar cuestiones sobre aspectos muy diversos del sistema durante el desarrollo del mismo y permite a los usuarios interaccionar con él y explorar sus posibilidades.

El uso de los prototipos en el desarrollo de sistemas software no se limita sólo a probar las

interacciones que los usuarios deben realizar, sino que son útiles también para otras actividades que se realizan durante el proceso, como por ejemplo su gran utilidad en la fase de recogida o análisis de requisitos en cuanto que amplía y mejora y la información necesaria para el desarrollo del sistema.

Propiedades:

- Son formidables herramientas de:

  - ✓ Comunicación entre todos los componentes del equipo de desarrollo y los usuarios.
  - ✓ Participación, para integrar activamente a los usuarios en el desarrollo.
  - ✓ Exposición de los mal entendidos entre los usuarios del software y los desarrolladores.

- Dan soporte a los diseñadores a la hora de escoger entre varias alternativas.
- Permiten a los diseñadores explorar diversos conceptos del diseño antes de establecer los definitivos.
- Permiten evaluar el sistema desde las primeras fases del desarrollo (facilitan la exploración de ideas sobre nuevos conceptos tecnológicos).
- Son el primer paso para que ideas abstractas sean concretas, visibles y testables.
- Fomentan la iteratividad.
- Mejoran la calidad y la completitud de las especificaciones funcionales del sistema.
- Se detectan los servicios que hacen falta.
- Se identifican servicios confusos.
- Permiten evaluar el sistema desde las primeras fases del desarrollo (facilitan la exploración de ideas sobre nuevos conceptos tecnológicos).
- Son herramientas de propósito general, pues sirven para comprobar la fiabilidad técnica de una idea, clarificar requisitos que quedaron "indeterminados" o ver cómo responde con el resto de la aplicación.

Desventajas:

- Costo de entrenamiento/capacitación en la herramienta.
- Costo de realizar el prototipo.
- Incompletitud (puede confundir a los usuarios, haciéndolos pensar que el producto final quedará como el prototipo, incompleto).

Prototipado

Prototipado, es una palabra de uso común en el ámbito de la Ingeniería de Software que se utiliza como traducción del anglicismo "*prototyping*" que viene a ser un sustantivo que aglutina el significado de la palabra prototipo con las diferentes herramientas y técnicas que permiten la producción de dichos prototipos.

- Se usa un prototipo para dar al usuario una idea concreta de lo que va a hacer el sistema.
- Se aplica cada vez más cuando la rapidez de desarrollo es esencial.
- Prototipado exploratorio: el prototipo inicial se refina progresivamente hasta convertirse en versión final.

- Prototipado desechable: de cada prototipo se extraen ideas buenas que se usan para hacer el siguiente, pero cada prototipo se desecha entero.

Categorías de técnicas

Las diferentes técnicas de realización de prototipos varían entre ellas en términos de:

- El costo y el esfuerzo de producir los prototipos y
- La fidelidad de dichos prototipos respecto al sistema final.

Será necesario, por tanto, valorar en cada momento cuál será la técnica más apropiada a utilizar en función del período del desarrollo en el que nos encontremos y de los objetivos a cumplir.

Las técnicas de prototipado suelen catalogarse en dos categorías básicas: Baja fidelidad (*low-fidelity o low-fi*) y alta fidelidad (*high-fidelity o hi-fi*).

- Los prototipos de baja fidelidad implementan aspectos generales del sistema sin entrar en detalles. Permiten abarcar un espectro mayor de la interacción a realizar.
- Con los prototipos de alta fidelidad se representan aspectos más precisos. Sirven, por ejemplo, para detallar el proceso interactivo global de una o varias tareas concretas.
- Los prototipos de baja fidelidad se caracterizan por ser económicos, rápidos de construir, rápidos de arreglar y no precisan de técnicos expertos.
- Los prototipos de alta fidelidad se caracterizan por el uso de herramientas especializadas de prototipado que ofrecen más detalle y precisión, por requerir de expertos que conozcan dichas herramientas, por ser más caros (tanto las herramientas como los expertos no son precisamente baratos), por necesitar mayor tiempo para implementar el prototipo y los cambios, por crear falsas expectativas (suelen hacer creer al usuario y/o cliente que el producto está más avanzado de lo que realmente está).

En la tabla 26, se pueden apreciar las características de los prototipos de alta y baja fidelidad.

Tabla 26: Características de los prototipos de alta y baja fidelidad

Categorías	***Ventajas***	***Desventajas***
Baja Fidelidad	• Costos de desarrollo pequeños. • De muy rápida creación. • Fácil de cambiar (cualquiera puede realizar los cambios). • Los usuarios, al ser conscientes de la facilidad de los cambios y del bajo costo económico, se sienten cómodos para opinar y proponer cambios. • Evaluación de múltiples conceptos de diseño. • Útil para el diseño general de las interfaces. • Útil para identificar requisitos.	• Limitado para la corrección de errores. • Especificaciones poco detalladas (para pasar a la codificación). • Su utilidad disminuye cuando los requisitos ya están bien establecidos. • Navegación y flujo de acciones limitadas.
Alta Fidelidad	• Funcionalidad completa de tareas. • Completamente interactivo. • Dirigido por el usuario. • Navegabilidad. • Aspecto semejante al sistema final. • Puede servir como especificación. • Puede servir como herramienta de marketing y para demostraciones de ventas.	• Elevados costos de desarrollo. • Requieren mucho tiempo de implementación. • Mayor dificultad de cambiar (cambios sólo realizables por el autor y requieren mayor tiempo). • Menor efectividad para la recolección de requisitos.

Dimensiones

La razón principal del uso de los prototipos es la reducción en costo y tiempo que supone su uso en la implementación del futuro sistema, esta reducción se puede conseguir o bien reduciendo el número de características o bien reduciendo el nivel de implementación de las funcionalidades de las características, esto define dos dimensiones denominadas prototipos horizontal y vertical.

- Prototipado vertical. El resultado de este tipo de prototipo es un sistema que tiene implementadas pocas características, pero sus funcionalidades están totalmente implementadas. Un prototipo vertical puede probar, por tanto, una parte limitada del sistema, pero puede ser probado en profundidad bajo circunstancias reales.
- Prototipado horizontal. Incluye toda la interfaz de todas las características del sistema, pero no contiene funcionalidad subyacente. Un prototipo horizontal es una simulación de la interfaz en la que no se puede realizar ningún trabajo real.

En la tabla 27, se pueden apreciar las distintas dimensiones para un prototipo y los rangos que pueden alcanzar.

Tabla 27: Rango de valores para las dimensiones

Enfoque de prototipado	***Nivel***	
Desechabilidad	Nula	Alta
Fidelidad	Baja	Alta
Implementabilidad	Nula	Alta
Digitalidad	Baja	Alta

Técnicas de prototipado

Maquetas:

Las maquetas son básicamente una sola página de representaciones estáticas del espacio de diseño. Se usan para mejorar el diseño y facilitar la comunicación entre los diseñadores, usuarios e implicados.

- Boceto. Los bocetos son maquetas rápidas, pequeñas para desarrollar un amplio espectro de ideas de diseño. Se usan típicamente en la primera etapa del diseño, muchas veces de que se haya acabado la fase de análisis de requisitos. La clave de los bocetos es su velocidad. Cada boceto no puede costar más de 15 o 20 segundos, de esta manera se pueden generar una gran cantidad de bocetos en poco tiempo.
- Maqueta de papel. Son representaciones de una mayor cualidad que los bocetos. Permite explorar el diseño de la página, aspectos estéticos. Permite una comprensión más realista de los límites del tamaño de la página, del espacio de diseño, identificar posibles dificultades en el proceso de diseño y comenzar la evaluación con los usuarios.
- Maqueta digital. El prototipo digital constituye un paso más en la elaboración de prototipos el cual es realizado con herramientas de diseño gráfico, siendo, por tanto más costosos y elaborados. Permiten afinar aspectos importantes del diseño cómo son los colores, los contrastes, las fuentes tipográficas, etc. que el prototipo de papel no proporciona.

Prototipo en papel:

Este tipo de prototipo se basa en la utilización de papel, tijeras, lápiz o instrumentos que se puedan utilizar para describir un diseño en un papel. Este método permite una gran velocidad y flexibilidad.

Escenarios:

Una historia de ficción con representación de personajes, sucesos, productos y entornos. Ayuda al diseñador a explorar ideas y las ramificaciones de decisiones de diseño en situaciones concretas. Es interesante pensar en diferentes escenarios para reflejar las distintas situaciones y puntos de vista diferentes.

Herramientas de diagramación:

- Narrativa. Una historia completa de la interacción hecha con la existente o con un diseño nuevo.
- *Flowchart*. Una representación gráfica de las series de acciones y decisiones extraídas de la narrativa
- Texto procedural. Una descripción paso a paso de las acciones del usuario y las respuestas del sistema
- *Storyboard*. Un s*toryboard* es una narración gráfica de una historia en cuadros consecutivos
- Elementos: Configuración (*setting*); Agentes o actores; Diagrama de secuencia de acciones y eventos
- Tipos: Escenario de tareas - Es una descripción del mundo del usuario tal como existe ahora. Escenario de futuro - Es una descripción del mundo del usuario en un futuro

En la tabla 28, se puede apreciar las características, los costos y el tiempo de desarrollo de las distintas técnicas de prototipado.

Tabla 28: Características, los costos y el tiempo de desarrollo

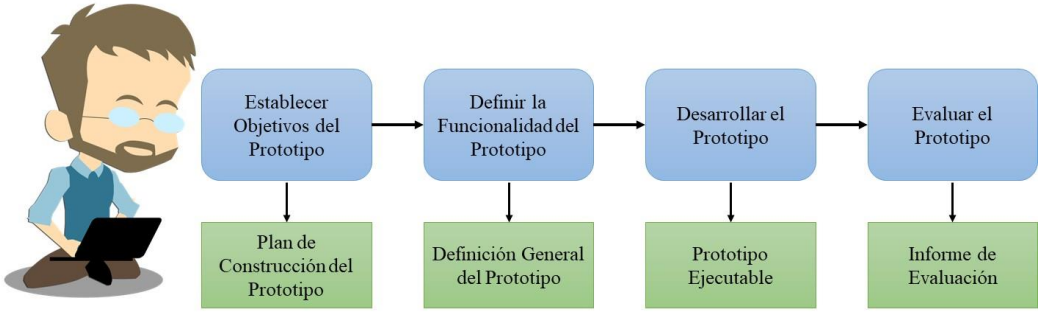
Técnica	***¿Qué se analiza?***	***Costos***	***Desarrollo***
Bocetos	• Primeras ideas	• Muy bajo	• Muy rápido
Maquetas	• Reproducir el uso • Reproducir las características físicas	• Relativamente bajo (aunque depende del material utilizado)	• No tan rápido
Maquetas digitales	• Visibilidad de las funciones (más que la propia funcionalidad) • Metodología de interacción (facilidad) • Disposición de los elementos interactivos de la interfaz	• Bajo	• No tan rápido
Prototipo en Papel	• Simplicidad, minimalismo • Visibilidad de las funciones (más que la propia funcionalidad). • Los cambios se pueden realizar muy rápidamente y sobre la marcha. • Si el diseño no funciona se puede reescribir las hojas erróneas o rediseñarlas y volver a probar la tarea a realizar. • Los usuarios o los actores se sienten más cómodos para poder realizar críticas al diseño debido a la sencillez del mismo por lo que no se sienten cohibidos a dar sus opiniones.	• El costo es muy reducido, necesitando únicamente los recursos humanos dedicados a la realización del prototipo.	• Rápido

Storyboards	• Reproduce el contexto. • Descripción del proceso de interacción. • Identificación y ubicación de los actores y objetos que intervienen en la interacción.	• Muy bajo	• Rápido
Escenarios	• Representación de casos o situaciones interactivas. • Entender el contexto y el porqué de la tarea. • Visibilidad de los actores y de los objetos que intervienen en la interacción. • Escenificación de posibilidades futuras o de acceso difícil.	• Alto	• Alto
Prototipos Software	• Navegabilidad. • Seguimiento de las tareas. • Globalidad del proceso interactivo. • Exploración de funcionalidades concretas. • Medidas de rendimiento. • Posibilidad de realizar evaluaciones por métricas.	• Medio	• Medio-Alto

Proceso de prototipado

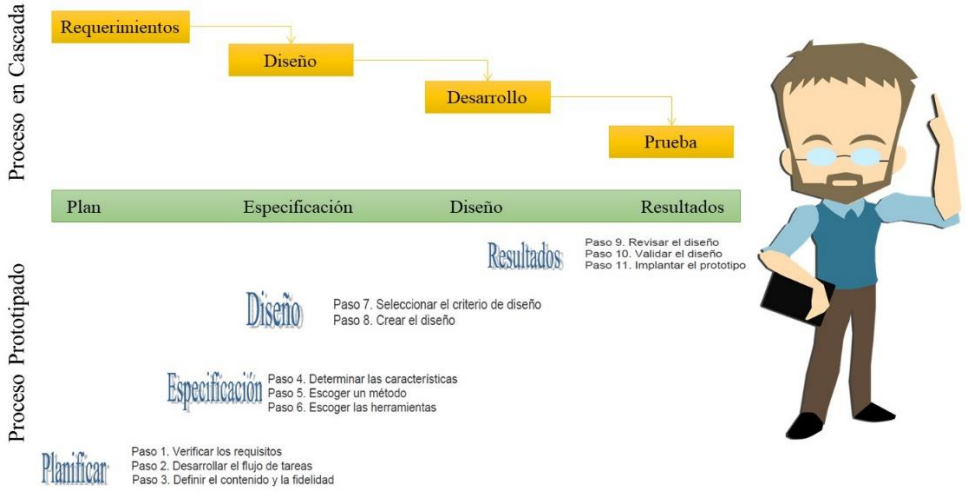
En la figura 25, se presenta el proceso de prototipado. Cabe destacar que las etapas son similares a las necesarias para construir un software, pero con el objetivo de lograr una aplicación de tipo prototipo.

Figura 25: Proceso de prototipado



Método de Arnowitz et al.

Figura 26: Método de Arnowitz, J., et al. (2007)



El método de Arnowitz, J., et al. (2007), se presenta en la figura 26. Este método está orientado a la construcción de aplicaciones, mediante un enfoque de prototipos, orientadas al comercio electrónico en la web.

LECTURA ADICIONAL RECOMENDADA

1. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.
2. Leiva, I., Villalobos, M. (2015). Método ágil híbrido para desarrollar software en dispositivos móviles. Ingeniare. Revista chilena de ingeniería, 23(3), 473-488. https://dx.doi.org/10.4067/S0718-33052015000300016
3. Abascal, J., Aedo, I., Cañas, J., Gea, M., Gil, A., Lorés, J., Martínez, A., Ortega, M., Valero, P., Vélez, M., "La interacción persona-ordenador", Editor: Jesús Lorés, ISBN: 84-607-2255-4, 2001.
4. Arnowitz, J., et al., Effective Prototyping for Software Makers, Morgan Kaufmann, Elsevier, Inc., 2007.
5. De Antonio, A., Gestión, control y garantía de la calidad del software, Apuntes, Universidad Politécnica de Madrid, año 2004.
6. Herrera, V., "Desarrollo de un Plan de Gestión de Mantenimiento de Software para el Departamento de Sistemas de la Universidad Politécnica Salesiana Basado en la norma ISO/IEC 14764:2006, Universidad Politécnica Salesiana, Tesis de pregrado, Cuenca, Ecuador, 2015.
7. Pressman, Roger s., Ingeniería de software, un enfoque práctico, sexta edición. Editorial McGraw Hill, México, año 2006.
8. Sommerville, Ian, Ingeniería de software, sétima edición, Editorial Pearson Addison Wesley, España, año 2005.

EJERCICIOS PROPUESTOS

4.1 Aseguramiento de la calidad

a. Aseguramiento

i. ¿Qué es la Garantía de la Calidad (GC)?
ii. ¿El equipo humano de GC es el mismo que el de Desarrollo?, ¿Por qué?
iii. ¿Cuáles son las áreas que caen bajo la responsabilidad del grupo de GC?
iv. ¿Cuáles son las principales tareas del grupo de GC?
v. Indique los diferentes tipos de actividades constructivas de GC, ejemplificando

b. Plan de Garantía de Calidad (PGC)

i. Explique el diagrama de la figura 13 que señala como construir un PGC
ii. Explique su propuesta PGC para el proyecto que usted dirige.

4.2 Fundamentos para ejecutar el proyecto

Ciclos de Vida

i. Defina el Ciclo de Vida del Software. Justifique la importancia de este concepto y la necesidad de su utilización.

ii. Explique la figura 14 del texto.
iii. Explique la representación genérica de Villalobos (figura 15).
iv. Complete la siguiente tabla:

Estilo de Ciclo de Vida	*Descripción*	*Fortalezas*	*Debilidades*
Secuencial – Lineal			
Secuencial-Cascada			
Secuencial-Cascada Sashimi			
Secuencial-Cascada con Subproyectos			
Iterativo – Incremental			
Iterativo-Evolutivo			
Iterativo-Prototipos			
Espiral			

v. Razone qué criterios deben guiar la elección de un modelo de ciclo de vida.
vi. Frente a la fabricación 'intuitiva' o 'artesanal' de un producto software ¿qué ventajas tiene el uso de las técnicas de ingeniería del producto software, vistas en la asignatura (en cuanto al uso de un ciclo de vida, el análisis, el diseño, la codificación, cómo se hacen las pruebas, la integración, etc.), respecto al producto final obtenido y el proceso de su desarrollo?
vii. Considere el desarrollo de un software cuyo dominio de aplicación es conocido, sus objetivos y requerimientos funcionales son estables y simples de comprender desde un principio, la tecnología a utilizar ya está predeterminada y es bien conocida por el equipo de desarrollo. ¿Qué tipo de modelo de ciclo de vida elegiría para el desarrollo de dicho software?.
viii. Una vez elegido el modelo de ciclo de vida, para el desarrollo del sistema planteado en el ejercicio anterior, ¿Qué etapas escogería para dicho modelo de ciclo de vida, teniendo en cuenta que el desarrollo lo realizan una o pocas personas?.
ix. Considere ahora el desarrollo de un sistema cuyo dominio de aplicación no es muy conocido por el equipo de desarrollo. En este caso, el cliente tampoco tiene muy claro qué es lo que quiere, de manera que los objetivos y requerimientos funcionales del sistema son inestables y difíciles de comprender. Además, el equipo de desarrollo va a utilizar una tecnología que le resulta completamente nueva. Indique qué modelo de ciclo de vida es más apropiado para esta situación.
x. Usted recientemente se ha incorporado a la empresa 'VillaSoft' como director/a de software. Dicha empresa lleva muchos años desarrollando software de gestión contable para pequeñas empresas y utilizando el ciclo de vida en cascada con éxito aceptable. Sin embargo, según su experiencia, usted piensa que el modelo con prototipo rápido es una forma bastante mejor para desarrollar software. Escriba un informe, dirigido al presidente de desarrollo de software, explicando por qué cree que la organización debería cambiar al uso del modelo con prototipo rápido. Recuerde que al presidente no le agrada los informes de más de una página y menos con faltas de ortografía.
xi. Explique los inconvenientes que tiene el utilizar un ciclo de vida clásico para desarrollar determinado producto en el seno de una organización que carece de experiencia en el desarrollo de este tipo de productos y en las tecnologías implicadas.
xii. ¿Cuál es la diferencia entre un modelo de ciclo de vida clásico y ágil?
xiii. ¿Cuáles son las características del desarrollo ágil?

4.3 Etapas de desarrollo de software

i. Enumere y explique las distintas etapas asociadas al desarrollo de software (las denominadas Fases en los Ciclos de Vida). Describa que se hace en cada etapa con un breve ejemplo.
ii. Investigue cómo le denominan los distintos autores a las distintas etapas asociadas al desarrollo de software completando la siguiente tabla:

FASE	*Autor 1*	*Autor 2*	*Autor 3*	*Autor 4*
Especificación de Requisitos				
Análisis				
Diseño				
Implementación				
Pruebas				
Mantenimiento				

Especificación de Requisitos

- Explique la definición de requisito software que presenta el estándar 610.12-1900, IEEE.
- Señale las distintas clasificaciones para los requisitos de software.
- Ejemplifique con un enunciado los distintos requisitos no funcionales que podría existir para un software, de acuerdo a la figura 20 del texto.
- Indique las distintas fuentes de requisitos que deben ser estudiadas al momento de una especificación de requisitos. Ejemplifique.
- Señale cuáles son los problemas típicos que se pueden encontrar a la hora de realizar una especificación de requisitos.
- Explique a que etapa de la actividad de construcción de un edificio corresponde la especificación de requisitos de software.
- Explique cada una de las características deseables de una especificación de requisitos software, dando ejemplos.

Característica	Explicación / Ejemplo
No ambigua	
Completa	
Correcta	
Comprensible	
Internamente consistente	
Externamente consistente	
Realizable	
No redundante	
A un nivel de detalle	
Precisa	
Concisa	
Independiente del Diseño	

Trazable	
Modificable	
Electrónicamente almacenada	
Anotada por relativa importancia	
Trazada	
Organizada	
Con referencias cruzadas	

- Explique cada una de las etapas del proceso de Gestión de Requisitos que se indican en la figura 24 del texto.
- Clasifique los siguientes requisitos:

Especificación	FUNCIONAL	NO FUNCIONAL		
		Del producto	Organizacional	Externo
El costo de implementación no deberá exceder de US$ 10,000.00				
El menú de navegación siempre deberá estar disponible al lado izquierdo.				
A cada usuario se le asignará un usuario del sistema y una clave, los cuales permitirán el ingreso de acuerdo un perfil determinado.				
Obligar al usuario a cambiar su contraseña cada 2 meses				
Permitir que el usuario pueda cambiar la contraseña de acuerdo a las políticas de seguridad de la organización.				
La implementación del desarrollo será desplegado en Internet.				
El sistema podrá subir la temperatura del agua por medio del regulador.				
El lenguaje de Programación a utilizar será Java.				
El tiempo promedio entre fallas estimado será de una vez cada 6 semanas.				
El sistema deberá mantener almacenado los errores originados por excepciones en el sistema (Log).				
El sistema deberá mantener almacenado el contenido histórico de todas las operaciones (Log).				
La aplicación se desarrollará en el Lenguaje de programación Java bajo la tecnología JAVA2EE. Con la IDE Eclipse.				
El sistema deberá considerar una arquitectura lógica basada en el patrón de diseño MVC.				
Para la gestión de base de datos se usara MySql.				
La aplicación Web deberá ser compatible con los navegadores Internet Explorer 7 y Mozilla Firefox 3.01				
Los usuarios podrán realizar reservas de libros, independientemente de si el libro deseado se encuentra o no prestado.				
El Servidor de aplicaciones será Apache Tomcat.				
El diseño de la interfaz deberá ser diseñada usando la herramienta Adobe Dreamweaver.				
No se podrán reservar libros de consulta.				
La resolución mínima para una buena visualización y ejecución del sistema será un tamaño de pantalla de 1024x768 píxel.				

Los reportes mostrarán el logotipo y nombre de la empresa.				
El sistema debe alinearse con la red implementada en la empresa y no deberá generar conflicto con las aplicaciones existentes.				
Un usuario podrá reservar un número indefinido de libros. Igualmente, múltiples usuarios podrán reservar un libro determinado.				
La implementación quedará concluida a fines de Julio de 2016.				
El sistema debe trabajar sobre cualquier computador que cuente con estos requerimientos mínimos con procesador Pentium IV o superior, 256 Mb de memoria RAM y disco duro de 20 GB.				
El Sistema deberá permitir el ingreso concurrente de por lo menos 20 usuarios distribuidos entre los diversos módulos del sistema a lo largo de las diversas oficinas con las que cuentan la compañía.				
El tiempo de respuesta del sistema para operaciones de ingreso o registro de información deberá ser como máximo 5 segundos de espera.				
El tiempo de transacción puede variar según la cantidad de datos y congestión en el sistema que pueda variar de entre 5 segundos a 10 segundos como máximo.				
El sistema debe permitir ser usado intuitivamente por cualquier usuario.				
La interfaz del usuario se diseñará de tal manera que le facilite el uso de la misma, sin necesidad de un soporte del área de sistemas. Esta tendrá que ser validada por el usuario.				
En caso de error del usuario el sistema informará claramente: el mensaje del error y la solución.				
El sistema estará disponible 24 horas al día, 7 días a la semana.				

Interfaz Gráfica de Usuario (IGU)

- ¿Qué es un Usuario?. De ejemplo de usuarios de tipos de software
- ¿Qué es la interfaz de un objeto?. De ejemplos
- ¿Qué es el *Affordance*?. De ejemplos de un buen *Affordance*
- De una definición para IGU
- Describa que es la Usabilidad y sus características
- Describa que es la Accesiblidad y sus características
- ¿Qué es un modelo mental (MM)?
- ¿Cuáles son las fuentes de los modelos mentales?
- Describa la relación entre MM e IGU
- ¿Qué es un modelo conceptual (MC)?
- Describa la relación entre MM y MC
- Describa que es una metáfora (M) en el contexto de las IGU
- Describa la relación entre M y MM
- Complete la siguiente tabla:

Área de aplicación	***Metáfora***	***Conocimiento familiar***
Sistemas operativos	Escritorio	

Entorno Orientado-objetos	Mundo físico	
Hipertexto	Tarjetas de notas	
Entorno de aprendizaje	Viaje	
Almacenamiento de archivos	Pila	
Entorno multimedios	Habitación (cada uno asociado con un medio/tarea)	
Trabajo colaborativo soportado por computador	Multi-agentes	

- ¿Qué es una metáfora visual (MV)?. De ejemplos
- ¿Qué es la intuición en el contexto de las IGU?
- ¿Qué es una metáfora compuesta?. De ejemplos
- Describa la técnica para construir una metáfora que se basa en la asociación con organizaciones (asociar estructuras, clases, objetos, atributos a nombres u operaciones; asociar procesos, algoritmos a verbos)
- Aplique la técnica anterior para los siguientes casos, diseñando un conjunto de metáforas que permitan la gestión de:

  - Una librería de imágenes electrónica
  - Un quiosco de publicaciones electrónicas (diarios, revistas, historietas, etc.)
  - Una boletería de eventos deportivos
  - Rutas verdes de una empresa de ferrocarriles
  - Un restaurante vegano
  - Una pastelería
  - Una clínica odontológica
  - Una Imprenta
  - Línea de buses interregional
  - Una asociación de futbol

- Explique cómo se aplica la técnica de revisiones (inspección) a la evaluación de IGU
- Explique cada uno de los factores de las 10 reglas heurísticas para evaluar IGU
- Explique cada una de las siguientes técnicas para construir prototipos de IGU: Maquetas, Prototipo de papel, *Storyboard* (historietas), Escenario, Prototipo de software

Pruebas

i. Explique la siguiente afirmación “Lo único que hay es un conjunto de aproximaciones metodológicas que facilitan y hacen más eficiente el proceso de prueba”
ii. Qué diferencia hay entre PRUEBA y DEPURACIÓN
iii. Indique el conjunto mínimo de pruebas que se deben realizar de acuerdo con el estándar IEEE 1012-1986
iv. Explique casa una de las pruebas según el estándar IEEE 1012-1986

Mantenimiento

- ¿Qué es el mantenimiento de software (MS)?

- ¿Qué relación tiene la cohesión y acoplamiento con el MS?
- Una práctica muy extendida para reutilizar software es "cortar y pegar" fragmentos de código. Esto puede llevar a que un trozo de código se encuentre repetido muchas veces en un programa. ¿Cómo afecta esto al mantenimiento del programa? ¿Cómo podrían solventarse los efectos negativos del código duplicado?
- Indique los tipos de MS. De ejemplos para cada uno de ellos
- Explique cada una de las técnicas asociadas al mantenimiento de código fuente de software
- Explique la figura 23 del texto
- Cuáles son las estrategias para re-documentar un software
- Explique el modelo para el mantenimiento de sistemas que ofrecen los estándares Std. IEEE 12207 – Std. ISO/IEC 14764
- Construya un Plan de Mantenimiento de Sistemas para la empresa VillaSoft que pueda ser aplicado a cada uno de sus proyectos de desarrollo de software. (use el ejemplo para su caso).

4.4 Métodos para desarrollar software

- Qué es un método
- Qué es un prototipo
- Qué es un marco de trabajo
- Qué es el prototipado
- Explique las etapas para desarrollar un prototipo
- Describa el modelo de proceso para construir prototipos de Arnowits et al.
- Describa cada una de las etapas del modelo de proceso de Arnowits et al.
- Aplique el modelo de proceso de Arnowits et al. a los software de gestión siguientes (para construir un prototipo de software):

    - Una librería de imágenes electrónica
    - Un quiosco de publicaciones electrónicas (diarios, revistas, historietas, etc.)
    - Una boletería de eventos deportivos
    - Rutas verdes de una empresa de ferrocarriles
    - Un restaurante vegano
    - Una pastelería
    - Una clínica odontológica
    - Una Imprenta
    - Línea de buses interregional
    - Una asociación de futbol

- Construya un prototipo de software, utilizando una herramienta de software como el *Balsamiq Mockups* u otra herramienta equivalente, para los diseños del problema anterior
- Simule la evaluación de sus prototipos de software mediante la técnica de inspección con la lista de chequeo de las 10 heurísticas

TRABAJO DE PROYECTO

Aplicación práctica 5

En el contexto del proceso de gestión del proyecto de desarrollo de software (asignado o elegido) corresponde, posterior a la aplicación de las actividades de las etapas de Inicio y Planificación, las actividades asociadas a la etapa de Ejecución. Entre las tareas a desarrollar son: Selección de un Ciclo de Vida para las actividades de Ingeniería, Selección de las Etapas de Ingeniería (Especificación de Requisitos, Análisis, Diseño, etc.), Selección de una o más herramientas que soporten el ciclo de vida y etapas de ingeniería. Sin embargo lo anterior, se podría seleccionar un Método que incluiría los dos primeros aspectos y eventualmente el tercero.
En esta oportunidad y dado que más adelante se seleccionará un método (por ejemplo, Jesús García Molina o RUP), solo se solicitará, el desarrollo de la etapa de Especificación de Requisitos que con seguridad será incluida por ellos u otros métodos.

Así, se requiere desarrollar la Especificación de Requisitos para la problemática seleccionada y asignada. ***El formato del respectivo informe será de acuerdo al estándar IEEE 830 1998 (puede usar plantillas) teniendo como título el informe "Especificación de Requisito para el Software XXX"***.

Como método de trabajo, se sugiere el siguiente enfoque:

Etapa	*Herramienta*
Educción	Entrevistas estructuradas, Revisión de Documentos, Observación de Campo
Análisis y Negociación	Descomposición, Abstracción y Revisión de información conjunta con el o los usuarios
Documentación	Uso del estándar IEEE 830-1998
Validación	Reuniones de Trabajo, Checklist (características de una buena ERS, entregada en clases)

El anexo del informe deberá incluir una o más entrevistas con sus respectivas respuestas (simuladas o realizadas), documentos que sean hallados y detalles logrados en una visita a las instalaciones. Estos aspectos serán especialmente valorados a la hora de evaluar el informe.

Aplicación práctica 6

Dentro de las actividades asociadas al diseño del software, se encuentra el "Diseño de una Interfaz Gráfica de Usuario". Para el caso en desarrollo, se le solicita realizar los siguientes trabajos. Cada alumno desarrollará lo solicitado según los problemas asignados.

1. Aplicar el método visto en clases, para crear una metáfora, cuyos pasos son: definición funcional (resumir los requisitos funcionales para el sistema de gestión asignado, en una lista tabulada con un orden de prioridad), generación de la metáfora (cuyos sub-pasos son: escoger los objetos que están implicados; Asociar un elemento visual a los objetos anteriores; Escoger los verbos asociados a las acciones que se pueden dar; Construir un elemento visual para el punto anterior).
2. Desarrollar un prototipo de software, aplicando el método de Arnowizt et al. Diseñe un conjunto de pantallas donde se "alojen" las metáforas desarrolladas, asociándolas con

"opciones de mayor nivel de abstracción" que el usuario requiere para hacer su trabajo. Debe ser desarrollado con Balsamiq Mockups u otra herramienta equivalente.
3. Para mejorar su propuesta se le solicita revisar a lo menos 3 software equivalente en la Web.
4. Evaluar el prototipo alcanzado, aplicando una la lista de chequeo con un conjunto de heurísticas basadas en los apuntes y el método de Arnowizt et al.

Aplicación práctica 7

En estos momentos se encuentra en la etapa de Ejecución de su proyecto, en particular actividad de Ingeniería de Proyecto, con el desarrollo del software definido para el proyecto de la empresa asignada. Sin embargo, como es de su conocimiento, es necesario elaborar un plan de mantenimiento ya que es muy probable que deba enfrentar alguno de los distintos tipos de mantenimiento.

Por lo anterior, se requiere la elaboración de un informe según el siguiente formato (sólo la sección de desarrollo, ya que el resto está determinado). Se le recuerda que el plan será genérico pero cada integrante del grupo de trabajo definido, se encargará de "contextualizarlo" para su software en desarrollo, para el cuál recién ha entregado un diseño mediante un prototipo.

III. DESARROLLO

Las secciones de estándar Std. IEEE 12207 – Std. ISO/IEC 14764.

Ayuda:

Para desarrollar el PM, deberá:

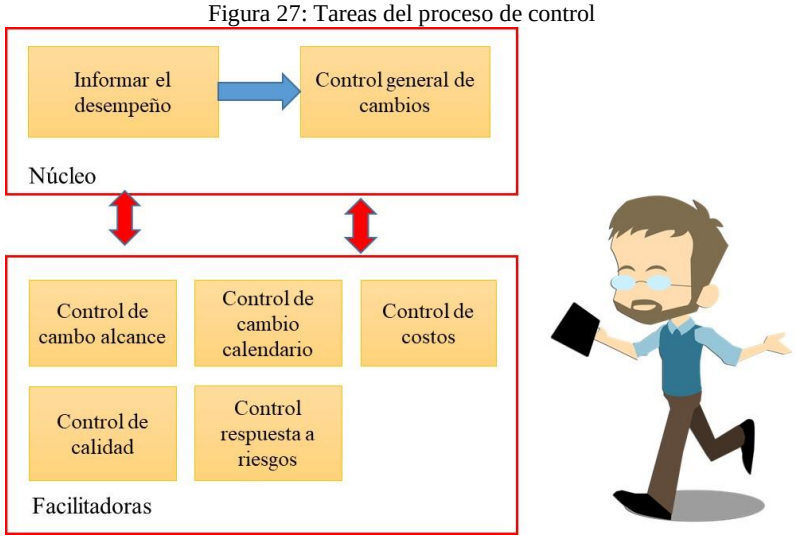
i. Analizar qué es un plan de mantenimiento (PM) (apuntes de clase)
ii. Saber cómo se desarrolla un PM
iii. Saber la estructura de un PM según el estándar Std. IEEE 12207 – Std. ISO/IEC 14764
iv. Desarrollar el PM para el proyecto definido, haciendo todas las suposiciones necesarias
v. Usar los ejemplos entregados por el profesor

CAPÍTULO 5: CONTROLAR UN PROYECTO

5.1 PROCESO DE CONTROL

En la figura 27, se muestra las tareas asociadas con el proceso de control de un proyecto.

Figura 27: Tareas del proceso de control



Tareas núcleo

- Informar desempeño: recolectar y diseminar información de desempeño; esto incluye informar el estado, medir el progreso y realizar proyecciones.
- Control General de cambios: coordinar los cambios entre las distintas partes del proyecto.

Tareas facilitadoras

- Control de cambios de alcance
- Control de cambios al calendario
- Control de costos: control de cambios al presupuesto
- Control de la calidad: controlar resultados específicos del proyecto para determinar si cumplen con estándares relevantes de calidad e identificar formas de eliminar causas de desempeño no satisfactorio.
- Control de respuestas al riesgo: responder a cambios en los riesgos a lo largo del proyecto

5.2 CONTROL DE CAMBIOS

¿Qué es la Gestión de Configuración del Software? (GCS)

Actividad constante aplicada durante todo el proceso de IS para identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo.

Comienza cuando se inicia el proyecto de desarrollo de software y termina sólo cuando el software

queda fuera de circulación.

Objetivos:

- Maximizar la productividad minimizando los errores
- Mejorar la facilidad para implantar cambios y reducir el esfuerzo para implementarlos
- Garantizar la calidad del software

Evita el caos porque:

- Se identifica el cambio
- Se controla
- Se garantiza que el cambio se implementa adecuadamente
- Se informa del cambio a todos los interesados

Mantenimiento software ≠ GCSw:

- Mantenimiento: Conjunto de actividades posteriores a la entrega del SW al cliente.
- Gestión de Configuración: Conjunto de actividades de seguimiento y control desde el inicio hasta el fin del proyecto + Mantenimiento.

Rol:

- ¿Qué papel juega la Gestión de Configuración dentro de un Proyecto de Desarrollo de Software? El éxito de un proyecto depende de la correcta ejecución de 4 tipos de funciones:

  - ✓ Gestión del Proyecto, que incluye fundamentalmente la Estimación, Planificación y Seguimiento del proyecto, y la Organización, Dirección y Gestión de Recursos Humanos.
  - ✓ Desarrollo Técnico: Actividades de Ingeniería del Software a lo largo de todo el ciclo de vida del producto (Especificación, Análisis, Diseño, Codificación)
  - ✓ Sistema de Calidad, que incluye las actividades de Validación (¿Estamos construyendo el producto correcto?), Verificación (¿Estamos construyendo el producto correctamente?) y Pruebas (¿Funciona el código?) y las actividades de Garantía de Calidad (Medidas encaminadas a asegurar que el producto se construye con unos determinados niveles de calidad).
  - ✓ Sistema de Gestión de Configuración, aunque también se suele considerar a la Gestión de Configuración como una actividad de Garantía de Calidad.

  Constituyendo las dos últimas una parte del Sistema de Control del Proyecto.

Actividades de la Gestión de Configuración del Software:

La definición estándar de Gestión de Configuración del Software, tal y como aparece en el estándar de IEEE, incluye las siguientes actividades:

- Identificación de la Configuración

- Control de Cambios en la Configuración
- Generación de Informes de Estado
- Auditoría de la Configuración

¿Qué es la Configuración del Software? ¿Y los Elementos de Configuración?:

- Al conjunto de toda la información y productos utilizados o producidos en un proyecto como resultado del proceso de ingeniería de software se le denomina configuración del software
- A cada uno de los componentes de la configuración del software se le va a llamar elemento de configuración del software (ECS). el ECS es la unidad de trabajo para la GCS
- El término configuración del software designa, por tanto, el conjunto de todos los elementos de configuración del software de un proyecto
- Se pueden considerar como ECS los siguientes componentes:

i. La especificación del sistema.
ii. El plan del proyecto software.
iii. La especificación de requisitos software.
iv. Un prototipo, ejecutable o en papel.
v. El diseño preliminar.
vi. El diseño detallado.
vii. El código fuente.
viii. Programas ejecutables.
ix. El manual de usuario.
x. El manual de operación e instalación.
xi. El plan de pruebas.
xii. Los casos de prueba ejecutados y los resultados registrados.
xiii. Los estándares y procedimientos de ingeniería de software utilizados.
xiv. Los informes de problemas.
xv. Las peticiones de mantenimiento.
xvi. Los productos hardware y software utilizados durante el desarrollo.
xvii. La documentación y manuales de los productos hardware y software utilizados durante el desarrollo.
xviii. Diseños de bases de datos.
xix. Contenidos de bases de datos.

Configuración del Software: El problema

- En cualquier caso, para cada proyecto concreto se debe determinar qué se va a considerar como ECS.
- Un ECS debe ser un elemento que se pueda definir y controlar de forma separada. Es decir, debe ser una unidad en sí mismo.
- Dependiendo de su tamaño, complejidad y necesidad de control y visibilidad sobre el mismo, puede requerir de su descomposición en varios ECS, aunque el sistema en su conjunto será a su vez un ECS.
- A medida que progresa el proceso de desarrollo, el número de ECS crece rápidamente. La

Especificación del Sistema da lugar al Plan del Proyecto y a la Especificación de Requisitos Software. A su vez estos dan lugar a otros documentos, etc. Si simplemente cada ECS produjera otros ECS, no habría prácticamente confusión.

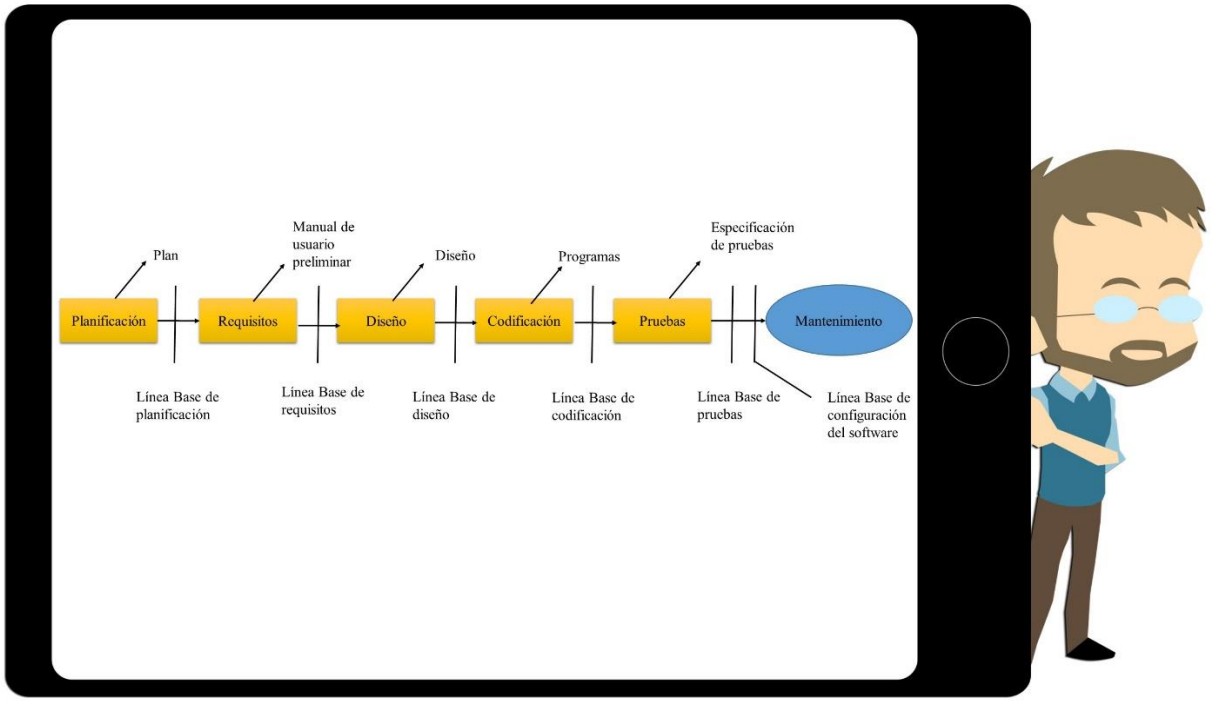
- El problema aparece cuando entra en juego la variable CAMBIO.

Línea base:

- Uno de los objetivos principales de la Gestión de Configuración va a ser el de gestionar los cambios que se producen en el sistema a lo largo de su ciclo de vida.
- Para controlar los cambios sin impedir los cambios justificados se utiliza el concepto de LÍNEA BASE.
- Se puede definir una línea base como un punto de referencia en el proceso de desarrollo del software que queda marcado por la aprobación de uno o varios Elementos de Configuración del Software, mediante una revisión técnica formal.
- También se puede definir como un elemento que ha sido revisado y aceptado, que va a servir como base para otros desarrollos posteriores y que solamente puede cambiarse a través de un proceso formal de control de cambios.
- La idea consiste en permitir cambios rápidos e informales sobre un Elemento de Configuración del Software antes de que se pase a formar parte de una línea base, pero en el momento en que se establece una línea base se debe aplicar un procedimiento formal para evaluar y verificar cada cambio.

En la figura 28, se muestra un ejemplo de las líneas base del proceso de desarrollo de software.

Figura 28: líneas base del proceso de desarrollo de software



Versiones, Revisiones, Variantes, Configuraciones Alternativas y *Releases*:

- Se puede definir una VERSIÓN como una instancia de un elemento de configuración, en un momento dado del proceso de desarrollo, que es almacenada en un repositorio, y que puede ser recuperada en cualquier momento para su uso o modificación.
- A las distintas versiones que aparecen en el tiempo, según se va avanzando en el desarrollo de un elemento, se les suele llamar también REVISIONES.
- Entre las distintas revisiones de un elemento de configuración se pueden establecer relaciones de sucesión temporal.
- Una representación posible para las diferentes revisiones de un elemento y sus relaciones de sucesión es mediante un GRAFO DE EVOLUCIÓN o grafo de revisión.
- La manera más fácil de crear una nueva revisión de un ECS es realizar una modificación sobre la revisión más reciente. De esta manera las revisiones van formando una cadena, a la que se suele llamar CADENA DE REVISIÓN.
- Cada revisión en la cadena es una actualización de, y viene a sustituir a la revisión anterior. Normalmente se trabajará sobre la última revisión de la cadena, que es la más actual, aunque las anteriores también deben ser accesibles.
- Cada una de las revisiones de un ECS se debe poder identificar de manera única, siendo común utilizar un esquema numérico, donde cada nueva versión recibe un número sucesivo.
- Existen herramientas de Control de Versiones que se encargan de la identificación de las distintas versiones de cada elemento, de mantener información acerca de las relaciones que hay entre ellas y de su almacenamiento en un repositorio.
- La Gestión de Configuración debe permitir también especificar y gestionar distintas VARIANTES de los elementos de configuración.
- Las variantes son versiones de un elemento de configuración que coexisten en un determinado momento y que se diferencian entre sí en ciertas características.
- Las variantes representan la necesidad de que un objeto satisfaga distintos requisitos al mismo tiempo.
- A diferencia con las revisiones, que son estrictamente secuenciales, y sólo existe una como revisión actual, las variantes se desarrollan en paralelo, y puede haber varias sobre las que se esté trabajando simultáneamente.
- Una variante no reemplaza a otra, como ocurre con las revisiones, sino que abre un nuevo camino de desarrollo. Las variantes se reconocen fácilmente en el grafo de evolución, puesto que aparecen como una ramificación de éste.
- El hecho de que se abran ramificaciones en el grafo de evolución hace que ya no se pueda hablar sencillamente de “la configuración” de un sistema.
- En vez de una única configuración vamos a tener un conjunto de CONFIGURACIONES ALTERNATIVAS.
- Cada configuración alternativa va a satisfacer las necesidades de un entorno particular o usuario, y se debe especificar mediante la selección de los ECS que la componen y de la versión adecuada de cada uno de ellos.
- Se suele llamar RELEASE a una configuración del sistema que se va a comercializar o entregar al cliente.

Identificación de la configuración

Esta tarea consiste en identificar y asignar nombres significativos y consistentes a todos y cada uno de los elementos que forman parte del producto software, en cada fase de su desarrollo, es decir, a cada uno de los Elementos de Configuración del Software.

La tarea de Identificación engloba varias actividades:

- Establecimiento de una jerarquía preliminar del producto software
- Selección de los elementos de configuración
- Definición de las relaciones en la configuración
- Definición de un Esquema de Identificación
- Definición y Establecimiento de líneas base
- Definición y Establecimiento de bibliotecas de software

Establecimiento de una jerarquía preliminar del producto software

- Se trata de obtener una primera visión de la estructura y los elementos que tendrá el sistema software.
- Esta jerarquía facilitará la ejecución de otras actividades posteriores de GC, como la selección de los elementos de configuración o la asignación de números de identificación a los documentos.

Selección de los Elementos de Configuración

- La selección de un número adecuado de ECS es muy importante.
- El tener demasiados puede provocar un número excesivamente elevado de especificaciones y documentos que al final resulta inmanejable.
- Otro problema es que mantener un EC es costoso, ya que puede requerir:

  - ✓ Especificación independiente
  - ✓ Plan de pruebas
  - ✓ Manuales independientes
  - ✓ Revisión por parte del usuario
  - ✓ Necesidad de aprobación de los cambios importantes por parte del usuario
  - ✓ Auditorías funcionales y físicas independientes
  - ✓ Número de identificación separado
  - ✓ Trazabilidad de su evolución

- Sin embargo, el tener pocos ECS puede hacer que no se tenga la suficiente visibilidad sobre el producto.
- Algunos criterios adicionales que se pueden utilizar para la selección de ECS son los siguientes:

  - ✓ Criticidad: Gravedad del impacto de un fallo en dicho componente.
  - ✓ Número de personas implicadas en su mantenimiento.
  - ✓ Complejidad de su interfaz: Las interfaces de un EC con otros deberían ser simples. Hay

que minimizar el acoplamiento entre ECs.
- ✓ Reutilización: Si el componente se va a diseñar especialmente para ser reutilizado.
- ✓ Tipo de tecnología: Si el componente incorpora nuevas tecnologías no utilizadas en otros componentes.

Definición de relaciones en la configuración

- Se puede considerar que los ECS son objetos, y están conectados con otros ECS mediante relaciones
- Esta información ayudará al personal de GCS a comprender dónde se sitúa un elemento con respecto al resto
- Hay que tener en cuenta que el personal de GCS suele ser externo al equipo de desarrollo y necesita este tipo de ayudas para poder asomarse a los productos y el proceso de desarrollo
- Algunas de las relaciones que puede ser interesante mantener son las siguientes:

  - ✓ Equivalencia: Por ejemplo, cuando un determinado ECS que es un programa está almacenado en tres lugares diferentes (un disco maestro, una copia de seguridad en cinta y el disquete del programador), pero todas las copias corresponden al mismo programa.
  - ✓ Composición: Por ejemplo, el ECS "especificación de diseño" estará compuesto de otros ECS, como el "modelo de datos" o el "diseño del módulo N", para cada uno de los módulos que componen el producto software.
  - ✓ Dependencia: Cualquier otro tipo de relaciones entre ECS, fundamentalmente en la documentación, y sobre todo para facilitar la Trazabilidad de los requisitos. Así, por ejemplo, el modelo de datos tiene dependencia con el Diagrama Conceptual.
  - ✓ Derivación: A partir de qué se ha originado algo. Por ejemplo, el código objeto del código fuente, o una determinada traza de ejecución de un determinado caso de prueba con un determinado programa ejecutable.
  - ✓ Sucesión: En la historia de cambios sobre un elemento desde una revisión a otra. Puede ser muy útil definir un Grafo de Evolución para cada ECS. Este grafo describe la historia de cambios de un objeto y su transición de unas versiones a otras.
  - ✓ Variante: Variación sobre un determinado elemento, con la misma funcionalidad, pero que, por ejemplo, funciona más rápido.

- Gracias a estas relaciones, si se lleva a cabo un cambio sobre un ECS, se podrá determinar fácilmente qué otros ECS pueden verse afectados.

Definición de un Esquema de Identificación

- Es necesario decidir también cuál es el método que se va a utilizar para identificar de forma unívoca cada Elemento de Configuración del Software. Dicho en otras palabras, es necesario establecer un Esquema de Identificación que permita etiquetar cada uno de los Elementos de Configuración del Software.
- Sea cual sea, el esquema de identificación utilizado debe proporcionar al menos la siguiente información:

  i. Número o código del Elemento de Configuración del Software.

ii. Nombre del ECS
iii. Descripción del ECS
iv. Autor/es del ECS
v. Fecha de creación
vi. Identificación del proyecto al que pertenece el Elemento de Configuración del Software.
vii. Identificación de la línea base a la que pertenece.
viii. Identificación de la fase y subfase en la que se creó.
ix. Tipo de Elemento de Configuración (documento, programa, elemento físico (cintas, discos, etc.),...).
x. Localización

- Por otro lado, el esquema de identificación debe permitir diferenciar entre las diferentes versiones de un mismo Elemento de Configuración del Software, puesto que los Elementos de Configuración del Software van a evolucionar a lo largo del ciclo de vida.
- Para ello, se suelen utilizar los siguientes campos:

xi. Número de versión
xii. Fecha de la versión

Definición y Establecimiento de líneas base

- El concepto clave para realizar esta actividad es el de Línea Base. Las líneas base se establecen en hitos previamente especificados a lo largo del proceso de desarrollo.
- Uno de los primeros pasos para poder efectuar la actividad de Identificación de la Configuración, por lo tanto, consiste en definir cuáles van a ser los hitos, dentro del proceso de desarrollo, en los que se va a establecer una línea base.
- Lo más corriente es establecer líneas base al finalizar determinadas fases del ciclo de vida de desarrollo, con dos objetivos:

  ✓ Identificar los resultados de las tareas realizadas durante la fase.
  ✓ Asegurar que se ha completado la fase.

- Aunque se pueden definir las líneas base con cualquier nivel de detalle, las líneas base más comunes son:

  ✓ Línea Base Funcional, que se establece al finalizar la fase de análisis y especificación de los requisitos del sistema, y comprende todos aquellos documentos en los que se define el problema a resolver, los costos del proyecto, el plan de tiempos, y los diferentes requisitos funcionales, de interoperatividad y de interfaz del sistema
  ✓ Línea Base de Distribución o Asignación de funciones, que se establece al finalizar la fase de análisis y especificación de requisitos software, y comprende toda la documentación que gobernará el desarrollo de cada uno de los componentes software que se han identificado en la especificación del sistema, y la asignación o reparto de las diferentes funciones entre los distintos componentes del sistema
  ✓ Línea Base de Diseño Preliminar, que se establece al finalizar la fase de diseño preliminar. Comprende todos aquellos documentos en los que se define la arquitectura del producto

software, así como el Plan de Pruebas

- ✓ Línea Base de Diseño, que se establece al finalizar la fase de diseño detallado. Comprende todos aquellos documentos que contienen el diseño detallado del software y el plan de implementación, y también la descripción de los casos de prueba
- ✓ Línea Base de Producto, que se establece al finalizar la fase de pruebas. Comprende los programas creados y todos aquellos documentos que contienen la información relativa a las pruebas realizadas.
- ✓ Línea Base de Operación, que se establece al finalizar la fase de implantación. Comprende los manuales de usuario, guías de operación y mantenimiento, manuales de formación, etc.

Definición y Establecimiento de bibliotecas de software

- Una biblioteca de software es una colección controlada de software y/o documentación relacionada cuyo objetivo es ayudar en el desarrollo, uso o mantenimiento del software.
- Facilitan la tarea de GCS, especialmente en cuanto al Control de Cambios y la Contabilidad de Estado.
- Se suelen establecer los siguientes tipos de bibliotecas de software o áreas:

  - ✓ Biblioteca de trabajo: Se establece al inicio del proyecto, y comprende el área de trabajo donde los analistas y diseñadores elaboran los documentos del proyecto y donde los programadores desarrollan el software, es decir, donde se realiza la codificación y pruebas unitarias. Una vez se han completado las revisiones o pruebas y el elemento de configuración en cuestión ha sido revisado y aprobado, se inicia la transferencia del ECS a la Biblioteca de Soporte al Proyecto. En esta biblioteca el control de cambios es informal.
  - ✓ Biblioteca de Integración. Es de esta biblioteca de donde se toman los ECS para su integración en ECS de nivel superior del sistema.
  - ✓ Biblioteca de Soporte al Proyecto: En esta biblioteca se almacenan los ECS aprobados y transferidos desde la Biblioteca de Trabajo o desde la Biblioteca de Integración. Cuando un elemento pasa a esta biblioteca, se encuentra sujeto a un control de cambios interno y semiformal.
  - ✓ Biblioteca de Producción: Está compuesta por la Biblioteca de trabajo, la de integración y la Biblioteca de Soporte al Proyecto.
  - ✓ Biblioteca maestra: Se usa para almacenar ECS liberados para su entrega al cliente o distribución en el mercado. Los elementos en la biblioteca maestra se encuentran sujetos a un control de cambios formal y estricto. Y normalmente esta biblioteca tiene fuertes restricciones de acceso para escritura, aunque normalmente no los tiene para lectura. En esta biblioteca se almacenan las *Releases* del sistema.
  - ✓ Repositorio de Software: Es la entidad en la que se archivan los ECS de un proyecto tras su cierre. Se transfieren a él desde la Biblioteca Maestra. Opcionalmente se puede identificar un segmento especial en el que se almacenarán los elementos reutilizables. Todo lo que se almacena en el repositorio debe estar adecuadamente identificado y catalogado, para facilitar su recuperación en caso de necesidad. Se supone que es un almacenamiento a largo plazo. Es central y común a todos los proyectos, mientras que la biblioteca de Producción y la Maestra son individuales para cada proyecto.
  - ✓ Biblioteca de *Backup*: También debe estar adecuadamente identificada, aunque su contenido no está sujeto a Gestión de Configuración (las copias contenidas en ella no están

catalogadas en los registros de Gestión de Configuración).

- Un ECS defectuoso podría volver en cualquier momento a la Biblioteca de Trabajo para ser modificado y generar una nueva versión.
- La tarea de bibliotecario suele estar encomendada a uno de los miembros del proyecto.
- Además de elegir el tipo de Bibliotecas de Software que se van a mantener, se deben definir los procedimientos para:

  - ✓ El establecimiento de una Biblioteca.
  - ✓ La introducción de elementos en la biblioteca.
  - ✓ El acceso a la biblioteca

Control de cambios

- Es la actividad de Gestión de Configuración más importante y su objetivo es proporcionar un mecanismo riguroso para controlar los cambios, partiendo de la base de que los cambios se van a producir. Normalmente combina procedimientos humanos y el uso de herramientas automáticas.
- Se pueden considerar fundamentalmente dos tipos de cambios:

  - ✓ Corrección de un defecto: Los clientes tienden a clasificar todos los cambios en esta categoría.
  - ✓ Mejora del sistema: Los programadores, sin embargo, los suelen clasificar aquí.

- Para solucionar el problema de determinar realmente de qué tipo es un cambio es importante la trazabilidad de los requisitos.
- Por lo general se establecen varios niveles de control de cambios:

  - ✓ Control de cambios informal: Antes de que el ECS pase a formar parte de una línea base, aquel que haya desarrollado el ECS podrá realizar cualquier cambio justificado sobre él.
  - ✓ Control de cambios al nivel del proyecto o semi-formal: Una vez que el ECS pasa la revisión técnica formal y se convierte en una línea base, para que el encargado del desarrollo pueda realizar un cambio debe recibir la aprobación de:

    - El director del proyecto, si es un cambio local
    - El Comité de Control de Cambios, si el cambio tiene algún impacto sobre otros Elementos de Configuración del Software

  - ✓ Control de cambios formal: Se suele adoptar una vez que se empieza a comercializar el producto, cuando se transfieren los ECS a la Biblioteca Maestra. Todo cambio deberá ser aprobado por el Comité de Control de Cambios.

- En un proceso formal o semi-formal, aparece una nueva figura en la Organización, el Comité de Control de Cambios.
- El Comité de Control de Cambios es una persona o grupo encargado de tomar las decisiones finales acerca del estado y la prioridad de las peticiones de cambio.

- Su obligación es tener una visión general del producto para poder evaluar el impacto de cada cambio en un determinado ECS sobre otros ECS, así como el impacto sobre la calidad del producto, su rendimiento, su fiabilidad, la visión que el cliente tiene del producto, etc.
- Esta labor se suele delegar en el Comité de Revisión del Diseño.
- Dependiendo del tamaño del proyecto y de la empresa estará integrado por una o varias personas.
- De todas formas, no sólo el Comité de Control de Cambios es responsable del Control de Cambios, también van a tener alguna responsabilidad:

  - ✓ Todos los miembros de un proyecto pueden solicitar cambios serán los encargados de realizar los cambios deben informar acerca del estado de los cambios que están realizando
  - ✓ El jefe de proyecto: se debe asegurar de que los procedimientos de Control de Cambios se siguen correctamente. Puede participar en la evaluación de los cambios, sobre todo en la determinación de impactos sobre otros elementos software, del costo del cambio y de su efecto sobre la programación del proyecto. Informar acerca del estado de los cambios.
  - ✓ El bibliotecario: debe controlar la realización de los cambios sobre las últimas versiones (o las adecuadas, en todo caso) transferirá los elementos a modificar desde la Biblioteca de Soporte del Proyecto a la Biblioteca de Trabajo. La Biblioteca Maestra no se modificará hasta que el cambio se haya incorporado y probado.

El proceso de control de cambios

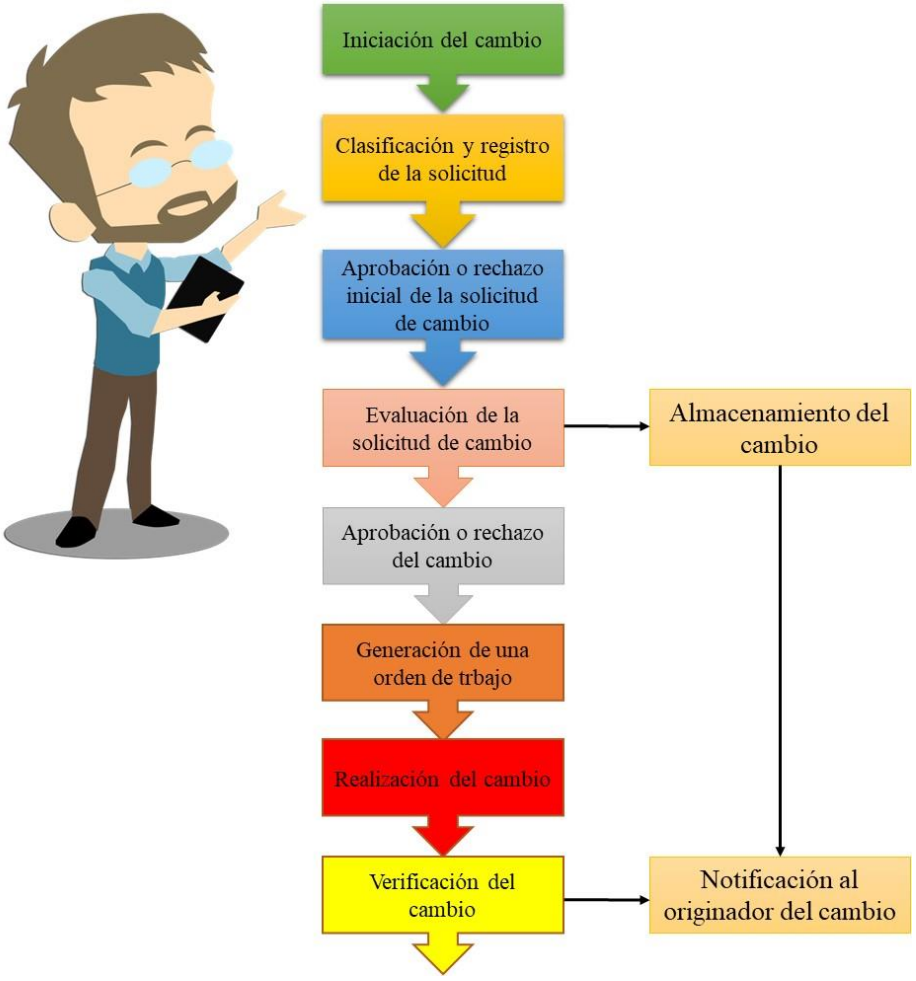
- No hay ningún estándar para el control informal o interno de los cambios, aunque sí hay algunas recomendaciones (IEEE STD 1042 *Guide to Software Configuration Management*).
- En cuanto al control de cambios formal, se puede estructurar de muchas formas.
- Las etapas típicas de un proceso formal, es decir, el proceso que habría que seguir para hacer un cambio sobre una línea base:

  - ✓ Iniciación del Cambio: se presenta una solicitud de cambio, que puede venir provocada por un problema que se ha detectado o por un cambio en los requisitos.
  - ✓ Clasificación y registro de la solicitud de cambio.
  - ✓ Aprobación o rechazo inicial de la solicitud de cambio. De ello suele ser responsable el Comité de Control de Cambios.
  - ✓ Evaluación de la solicitud de cambio, si ha sido aprobada, para calcular el esfuerzo técnico, los posibles efectos secundarios, el impacto global sobre otras funciones del sistema y el costo estimado del cambio. Como resultado se obtiene un Informe de Cambio.
  - ✓ Se presenta el Informe de Cambio al Comité de Control de Cambios. Si se considera que el cambio es beneficioso se genera una Orden de Cambio (también llamada Orden de Cambio de Ingeniería), que describe el cambio a realizar, las restricciones que se deben respetar y los criterios de revisión y de auditoría. Esta Orden de Cambio es asignada a alguno de los ingenieros de software para que se encargue de llevarlo a cabo. En este momento, el objeto a cambiar se da de baja en la Biblioteca de Soporte al Proyecto.
  - ✓ Se realiza el cambio, entrando en un proceso de seguimiento y control.
  - ✓ Una vez finalizado el cambio, se certifica, mediante una revisión, que se ha efectuado correctamente el cambio y con ello se ha corregido el problema detectado o bien se han satisfecho los requisitos modificados. El objeto se devuelve a la Biblioteca de Soporte al

Proyecto.
- ✓ Se notifica el resultado al originador del cambio.

En la figura 29, se muestra el proceso formal, es decir, el proceso que habría que seguir para hacer un cambio sobre una línea base.

Figura 29: Proceso formal para el cambio en la línea base



- Mecanismo para aprobar o rechazar las solicitudes de cambio. Algunos criterios que se pueden tener en cuenta para tomar la decisión de aprobar o rechazar las solicitudes de cambio son los siguientes:

  - ✓ valor del cambio para el proyecto/organización
  - ✓ retorno de la inversión
  - ✓ tamaño
  - ✓ complejidad
  - ✓ impacto sobre el rendimiento del producto (uso de memoria y CPU)
  - ✓ recursos disponibles para efectuar el cambio (humanos y materiales)
  - ✓ relación con otros cambios ya aprobados y en progreso

- ✓ tiempo estimado para completar el cambio
- ✓ relación con las políticas de la empresa (satisfacción del cliente, competitividad, etc.)

- Seguimiento de los cambios. Gestión de Problemas

  - ✓ Una vez aprobado un cambio debido a un problema se debe realizar un seguimiento del mismo. A este proceso se le llama Gestión de Problemas.
  - ✓ La Gestión de Problemas se considera una actividad complementaria a la de Control de Cambios, que consiste en gestionar la evolución de los problemas detectados sobre el software desarrollado, tanto aquellos que se detectan en la fase de pruebas como los informes de problemas que llegan del usuario.
  - ✓ Conlleva tareas como:

    - Admisión y registro de notificaciones de problemas (via llamadas telefónicas, correo electrónico, herramienta específica)
    - Asignación del problema a una persona responsable.
    - Asociación de información al problema y mantenimiento de la misma en una Base de Datos de Problemas.
    - Monitorización del estado del problema.
    - Registro de actividades efectuadas para resolver un problema.
    - Generación de informes acerca de los problemas.
    - Resolución de interrogaciones sobre la Base de Datos de Problemas.
    - Análisis estadísticos acerca de los problemas, como por ejemplo el estudio de correlaciones entre problemas y componentes.

  - ✓ Algunos de los datos que puede resultar interesante almacenar acerca de los problemas son:

    - descripción
    - severidad
    - urgencia o prioridad
    - causa del problema (omisiones en el análisis, error en la documentación de entrada, falta de experiencia,...)
    - solución al problema
    - módulos afectados
    - persona que lo notificó
    - persona responsable
    - fechas de notificación, resolución, etc.
    - fase/etapa en la que se originó el problema
    - fase/etapa en la que se detectó el problema

Generación de informes de estado

- El objetivo es mantener a los usuarios, a los gestores y a los desarrolladores al tanto del estado de la configuración y su evolución. En definitiva, pretende dar respuesta a la pregunta "¿Qué ocurrió?", y también a la pregunta "¿Cuándo ocurrió?".
- Ayuda también a mejorar los problemas de comunicación entre los participantes en un

proyecto.

- Esto se va a conseguir registrando toda la información necesaria acerca de lo que va ocurriendo y generando los informes necesarios. Esta tarea implica, por tanto, la realización de tres actividades básicas:

  - ✓ Captura de la información
  - ✓ Almacenamiento de la información
  - ✓ Generación de informes

- Así pues, los productos de esta actividad son fundamentalmente de dos categorías:

  - ✓ Registros.
  - ✓ Informes.

- ¿Por qué es importante esta tarea?

  - ✓ Para mantener la continuidad del proyecto. Se trata de permitir que el proyecto siga adelante cuando, por ejemplo, el jefe de proyecto deja la empresa.
  - ✓ Para evitar la duplicidad del trabajo. Si no se guarda información acerca de lo que ya se ha hecho, se puede estar repitiendo el trabajo ya hecho.
  - ✓ Para evitar caer en los mismos errores una y otra vez.
  - ✓ Para ser capaces de repetir aquello que salió bien.
  - ✓ Puede ayudar a encontrar las causas de un fallo.

- Registros. Algunos ejemplos de los tipos de registros que pueden mantenerse son los siguientes:

  a. Registro de elementos de configuración: Conteniendo toda la información relativa a los diferentes elementos de configuración.
  b. Registro de Líneas base: Conteniendo toda la información relativa a cada línea base: Nombre, Fecha de establecimiento, Elementos de configuración que la componen
  c. Registro de Solicitudes de cambios: El tipo de información que se suele mantener acerca de cada solicitud de cambio es la recogida a través del formulario de Solicitud de Cambio, incluyendo: código de solicitud, información acerca del solicitante, descripción del cambio, la documentación que apoya la petición de cambio, por ejemplo, una referencia a un Informe de Incidencia, la resolución o disposición acerca del cambio (aprobado, aplazado, denegado,...)
  d. Registro de Cambios: El tipo de información que se suele mantener acerca de cada cambio es la recogida a través del Informe de Cambio, la Orden de Cambio, el proceso de Gestión de Problemas, etc. Puede contener información acerca de:

     i. Solicitud del cambio a la que corresponde
     ii. Evaluación del cambio: Costo, Esfuerzo, Tiempo, Soluciones alternativas
     iii. Plan de implementación del cambio.
     iv. Restricciones y criterios de revisión
     v. Impacto sobre la configuración: Líneas base afectadas, Elementos de configuración afectados, Versiones afectadas

vi. Historia del cambio: Puesto que un cambio es algo que evoluciona, es necesario mantener una historia de cada cambio. En cuanto a los datos que se deben mantener en la historia de un cambio, se pueden considerar: Fecha de la solicitud de cambio, Fecha de aprobación del cambio, Fecha de rechazo del cambio, Fecha de cancelación del cambio, Fecha de implementación del cambio, Fecha de cierre del cambio, etc.

e. Registro de Incidencias: El tipo de información que se suele mantener acerca de cada incidencia es la recogida a través del Informe de Incidencia, del tipo:

i. Información del incidente
ii. Resultado de la Incidencia: Disposición del CCC, Número de la solicitud de cambio a la que dio lugar (si es aplicable), Número de Formulario de Seguimiento de Documentación (si es aplicable), Número de Notificación de Cambio asignada (si es aplicable)
iii. Historia: Fecha de la incidencia, Fecha de resolución acerca de la incidencia, Fecha de cierre de la incidencia

f. Registro de Modificaciones del Código: Puede contener información del tipo:

- ✓ Número de identificación de la modificación
- ✓ Descripción de la modificación
- ✓ Número de notificación de cambio a la que corresponde (si es aplicable)
- ✓ Número de solicitud de cambio a la que corresponde (si es aplicable)
- ✓ Nombre de los módulos afectados
- ✓ Versiones modificadas
- ✓ Persona responsable de la modificación
- ✓ Fecha de inicio
- ✓ Fecha de terminación
- ✓ etc.

g. Registro de Modificaciones sobre bases de datos:

- ✓ Número de identificación de la modificación
- ✓ Descripción de la modificación
- ✓ Número de notificación de cambio a la que corresponde (si es aplicable)
- ✓ Número de solicitud de cambio a la que corresponde (si es aplicable)
- ✓ Base de Datos modificada
- ✓ Número de versión modificada
- ✓ Registros modificados
- ✓ Persona responsable de la modificación
- ✓ Fecha de inicio
- ✓ Fecha de terminación
- ✓ etc.

h. Registro de Modificaciones sobre documentación:

- ✓ Número de identificación de la modificación.
- ✓ Descripción de la modificación
- ✓ Número de formulario de seguimiento de documentación al que corresponde
- ✓ Documento modificado
- ✓ Número de versión modificada
- ✓ Persona responsable de la modificación
- ✓ Fecha de inicio
- ✓ Fecha de terminación
- ✓ etc.

i. Registro de *Releases* y Variantes: Su objetivo es describir la composición y estado de una versión liberada del producto:

- ✓ Código de *release* o variante
- ✓ Fecha de liberación
- ✓ Elementos de configuración que la componen
- ✓ Versión de los elementos de configuración que la componen
- ✓ Medio en el que se encuentran
- ✓ Diferencias con la *release* anterior: Cambios incorporados, Cambios pendientes

j. Registro de Instalaciones: Su objetivo es mantener información acerca de todos los lugares en los que se ha instalado un producto software. Puede contener información del tipo:

- ✓ Identificación del producto
- ✓ Lugar en el que se ha instalado
- ✓ Fecha de la instalación
- ✓ *Release* instalada

k. Actas de las reuniones del Comité de Control de Cambios:

i. Fecha
ii. Lista de miembros asistentes
iii. Propósito de la reunión
iv. Acciones del CCC: ECS etiquetados; Líneas base revisadas/cambiadas; Disposición de Solicitudes de Cambio, Informes de Incidencias, Notificaciones de Cambio, Formularios de Seguimiento de Documentación, Informes de Cambios, etc; Líneas base aprobadas
v. Resultados de las auditorías: Deficiencias detectadas, Plan de resolución de las deficiencias encontradas, Recomendaciones

- Informes

- ✓ En cuanto a los informes, podemos distinguir dos tipos: Planificados, Bajo demanda.
- ✓ En cualquier caso, al comienzo de cada proyecto será necesario decidir qué tipo de registros se van a mantener y qué tipo de informes se van a generar y para quién.

a. Informe de estado de los cambios: Es un resumen del estado en que se encuentran todas las solicitudes de cambio registradas durante un determinado período de tiempo.
b. Inventario de elementos de configuración, para ofrecer visibilidad sobre el contenido de las bibliotecas de proyecto.
c. Informe de incidencias: Es un resumen del estado en que se encuentran todas las incidencias originadas durante un determinado período de tiempo y las acciones a las que han dado lugar.
d. Informe de modificaciones: Es un resumen de las modificaciones que se han efectuado en el producto software durante un determinado período de tiempo.
e. Informe de diferencias entre versiones: Resumen de las diferencias entre las sucesivas versiones de un elemento de configuración.

Auditoría de la Configuración

- Una auditoría es una verificación independiente de un trabajo o del resultado de un trabajo o grupo de trabajos para evaluar su conformidad respecto de especificaciones, estándares, acuerdos contractuales u otros criterios.
- La auditoría de la Configuración es la forma de comprobar que efectivamente el producto que se está construyendo es lo que pretende ser.
- Esta función a veces se considera fuera de la Gestión de Configuración y dentro de la Garantía de Calidad. También tiene relación con las actividades de Validación y Verificación. En realidad es un punto de intersección entre todas ellas.
- Es la actividad de GCS más costosa. Requiere de personal experimentado, y con un gran conocimiento del proceso de desarrollo. Sin embargo, debe ser realizada por personal ajeno al equipo de desarrollo técnico para mantener la objetividad de la auditoría.
- Se pueden diferenciar tres tipos de actividades:

  - ✓ Revisiones de fase: Se realizan al finalizar cada fase del desarrollo y su objetivo es examinar los productos de dicha fase. Las revisiones propias de la Gestión de configuración son aquellas en las que se establecerán las líneas base. El objetivo de esta revisión es descubrir problemas, no comprobar que todo está bien. Hay que ser capaz de desenmascarar los problemas ocultos y sutiles, no sólo los que son obvios.
  - ✓ Revisiones de cambios: Se realizan para comprobar que los cambios aprobados sobre una línea base se han realizado correctamente.
  - ✓ Auditorías: Se realizan al final del proceso de desarrollo de software y su objetivo es examinar el producto en su conjunto.

- Revisiones:

  - ✓ Las revisiones se deben realizar de forma continua, durante todo el proceso de desarrollo, y no sólo al finalizar éste, cuando los problemas ya no tienen solución.
  - ✓ La tarea de revisión implica tres tipos de funciones:

    - Verificar que la configuración actual del software se corresponde con lo que era en fases anteriores. Debe haber correspondencia y trazabilidad entre los elementos de configuración que aparecen en una línea base y los que aparecen en la línea base que la

preceden y que la siguen. La verificación se realiza con respecto a la línea base precedente.
- Validar que la configuración actual del software satisface la función que se esperaba del producto en cada hito del proceso de desarrollo. La validación se realiza con respecto a los requisitos del sistema.
- Valorar si una determinada línea base, teniendo en cuenta los resultados de la verificación y validación, y otro tipo de comprobaciones, se debe considerar aceptable o no.

✓ El papel que juegan las revisiones en el proceso de Gestión de Configuración es el siguiente:

i. Los productos generados durante el proceso de desarrollo se agrupan, al llegar determinados hitos, en una "línea base pendiente de aprobación".
ii. Tiene lugar la revisión de fase
iii. Si en la revisión se encuentran deficiencias en los ECS que componen la línea base, se generan los correspondientes Informes de Problemas, o Informes de Discrepancias, y se entregan al CCC, el cual recomienda ciertos Cambios sobre la Configuración.
iv. Una vez implementados los cambios propuestos, se pasa a una nueva revisión de cambios en la que se comprueba si los cambios se han implementado correctamente.
v. Si en la revisión no se encuentra ninguna deficiencia, se declara "aceptable" la "línea base pendiente de aprobación". Si el CCC está de acuerdo con la resolución de los revisores, la línea base se aprueba.

Auditorías:

- Se suelen distinguir dos tipos de auditorías de configuración:

✓ Auditoría Funcional: Cuyo objetivo es comprobar que se han completado todos los tests necesarios para el Elemento de Configuración auditado, y que, teniendo en cuenta los resultados obtenidos en los *tests*, se puede afirmar que el Elemento de Configuración satisface los requisitos que se impusieron sobre él.
✓ Auditoría Física: Cuyo objetivo es verificar la adecuación, completitud y precisión de la documentación que constituye las líneas base de diseño y de producto. Se trata de asegurar que representa el software que se ha codificado y probado. Tras la auditoría física se establece la línea base de Producto. Tiene lugar inmediatamente después de haberse superado la auditoría Funcional.
✓ Revisión Formal de Certificación: Cuyo objetivo es certificar que el Elemento de Configuración del Software se comporta correctamente una vez que éste se encuentra en su entorno operativo.

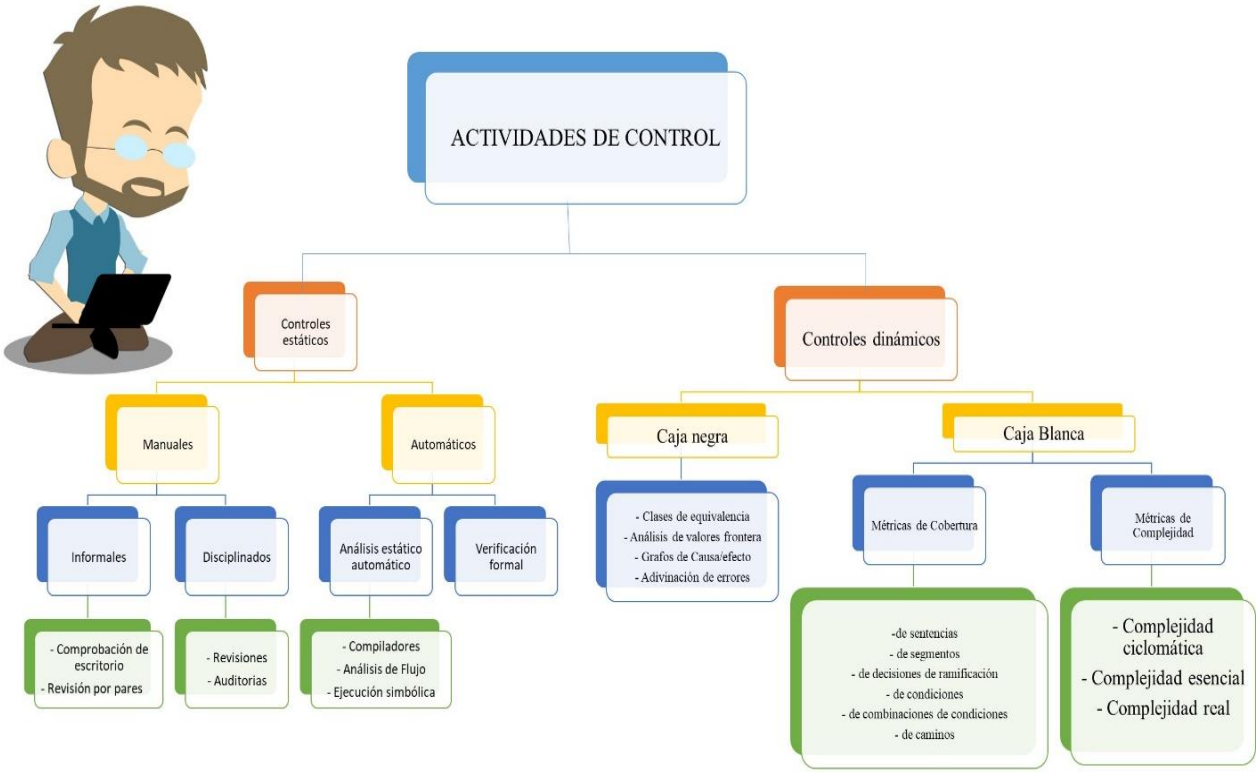
5.3 CONTROL DE CALIDAD

El objetivo de las actividades de Control de Calidad es comprobar si un producto posee o no posee una determinada característica de calidad en el grado requerido. Cuando un producto no posee una determinada característica de calidad se dice que tiene un defecto. Por lo tanto, se puede decir

también que el objetivo del Control de Calidad es identificar defectos en el producto y corregirlos. Se pueden clasificar las actividades de control de calidad en dos categorías: controles estáticos y controles dinámicos. Los primeros analizan el objeto sin necesidad de ejecutarlo mientras que los segundos requieren la ejecución del objeto que está siendo probado.

En la figura 30, se muestra los tipos de actividades de control.

Figura 30: tipos de actividades de control



Controles estáticos manuales disciplinados

Las revisiones y auditorías son la evolución natural de la Comprobación de Escritorio, pero a diferencia de aquélla pasan a ser técnicas de grupo. Su misión principal es conseguir que la responsabilidad del control de calidad no recaiga sólo sobre el propio desarrollador.

Auditorías

- Una auditoría consiste en realizar una investigación para determinar:

  ✓ El grado de cumplimiento y la adecuación de los procedimientos, instrucciones, especificaciones, códigos, estándares u otros requisitos de tipo contractual, establecidos y aplicables.
  ✓ La efectividad y adecuación de la implementación realizada.

- Se pueden considerar tres tipos de auditorías:

- ✓ Auditoría del producto: El objetivo es cuantificar el grado de conformidad del producto con las características requeridas. Las auditorías del producto software más comunes son la auditoría Funcional y la auditoría Física.
- ✓ Auditoría del proceso: El objetivo es evaluar el proceso de desarrollo o de gestión, y evaluar su completitud y efectividad, determinando dónde se puede mejorar. En el desarrollo de software se suelen realizar dos tipos de auditorías del proceso:

  - Auditorías de proyecto: cuyo objetivo es evaluar la productividad y eficacia del equipo que trabaja en un proyecto así como la efectividad de los métodos y herramientas utilizados.
  - Auditorías de gestión de proyecto: cuyo objetivo es evaluar la efectividad de las prácticas de gestión realizadas y la organización del proyecto.

- ✓ Auditoría del sistema de calidad: El objetivo es evaluar la completitud y efectividad del propio sistema de calidad establecido.

- El procedimiento habitual para realizar una auditoría consta de los siguientes pasos:

1. Planificación: Consiste en definir los objetivos de la auditoría y su alcance. En esta etapa se elabora un Plan de Auditoría, que debería dar respuesta a cuestiones del tipo de las siguientes:

   - ✓ ¿Por qué se realiza la auditoría? Puede ser una auditoría de rutina o puede realizarse para resolver problemas concretos.
   - ✓ ¿Para qué se realiza? para mejorar, para conseguir una certificación,...
   - ✓ ¿Cuál es el producto que va a ser auditado?
   - ✓ ¿Qué resultados se esperan de la auditoría? En principio, una auditoría debería identificar situaciones problemáticas, tales como desviaciones del estado actual con respecto al estado deseado, y sugerir posibles soluciones o mejoras.
   - ✓ ¿Cómo y dónde se van a utilizar los resultados de la auditoría?
   - ✓ ¿Quiénes son los responsables de llevarla a cabo?
   - ✓ ¿De qué forma se va a llevar a cabo? Incluyendo una especificación de los datos que se van a recoger y de qué forma se van a recoger.
   - ✓ ¿Cuándo se va a realizar?

2. Llevar a cabo la investigación. Por lo general la auditoría se inicia con una reunión de apertura de la investigación, y se lleva a cabo mediante entrevistas y revisiones en las que se recopilan datos.

3. Analizar los datos recogidos. Por lo general el equipo de auditores debe hacer frente a cantidades ingentes de datos, de entre los cuales resulta complicado seleccionar los datos relevantes, por lo que se suelen utilizar técnicas de análisis estadístico. A continuación se realiza una evaluación en paralelo de los resultados por un grupo de evaluadores, se comparan las conclusiones obtenidas y se estudian las causas de las desviaciones significativas.

4. Sugerir soluciones a los problemas encontrados y posibles mejoras.

5. Elaborar y presentar un informe de resultados.

Revisiones

Se puede definir una revisión como una reunión formal en la que se presenta el estado actual de los resultados de un proyecto a un usuario, cliente u otro tipo de persona interesada, y se realiza un análisis estructurado de los mismos.

Uno de los objetivos fundamentales de las revisiones técnicas es ofrecer a los gestores información fiable acerca de los aspectos técnicos del proceso de desarrollo de software, de la misma forma que les llega información fiable acerca de los costes y la programación del trabajo, para que con esta información puedan tomar decisiones adecuadas para dirigir con éxito el proyecto.

Con las revisiones se consigue que el peso de la evaluación técnica no recaiga sobre las mismas personas involucradas en la producción del software, que por la posición que ocupan no pueden ser totalmente objetivas, sino en otras personas técnicamente competentes y objetivas.

Las revisiones son, hoy en día, el único método de control de calidad eficaz en las fases iniciales del desarrollo a la hora de identificar desviaciones con respecto a las especificaciones de calidad. Las revisiones redundan en una mejora directa de la calidad del objeto que se examina y provocan, indirectamente, una mejora de la calidad del proceso de desarrollo, al facilitar la comunicación entre los miembros del equipo de desarrollo. Al mismo tiempo facilitan el control del coste y el tiempo.

Las diferencias más importantes entre las revisiones y las auditorías son las siguientes:

- Las revisiones se llevan a cabo desde las primeras fases del desarrollo, mientras que las auditorías se llevan a cabo en las fases finales.
- El objetivo de las revisiones es detectar defectos, mientras que el objetivo de las auditorías es certificar conformidad e identificar desviaciones.

Tipos de revisiones

Hay dos tipos fundamentales de revisiones: las inspecciones y los *walkthrough*. La diferencia entre ellos está en la forma en que se desarrolla la reunión de revisión.

- Inspecciones: En las que los participantes van leyendo el documento, paso a paso, guiados por el autor del mismo, y comprobando en cada paso el cumplimiento de los criterios de una lista de comprobación.
- *Walkthrough* (visita guiada): En las que se demuestra la funcionalidad del objeto revisado mediante la simulación de su funcionamiento con casos de prueba y ejemplos. Se introducen al objeto los casos de prueba y se van registrando los resultados intermedios.

Fases en una inspección

Una inspección consta de los siguientes pasos:

1. Planificación: La preparación comienza con la selección de los participantes. Debe designarse un coordinador o moderador; un secretario; un presentador, de entre los productores del objeto que se revisa; y otros revisores, que pueden ser desarrolladores, representantes de los usuarios, revisores externos; y posiblemente otros.

   El coordinador es responsable de la planificación de la inspección, de moderar la reunión (mantener el orden, mantener el foco de la revisión, asegurar que se cubren todos los aspectos necesarios), de preparar el informe final y de realizar el seguimiento y evaluación de las acciones pendientes.

   El secretario es responsable de anotar los elementos de interés (defectos y anomalías descubiertas, acciones pendientes) y ayudar al coordinador en la preparación del informe final.

   En la planificación es también necesario determinar:

   - ✓ Los objetivos de la inspección
   - ✓ Los criterios de finalización de la inspección
   - ✓ El lugar y la fecha para la inspección
   - ✓ La disponibilidad de todos los participantes
   - ✓ La agenda de la reunión

2. Orientación inicial: Es recomendable realizar una reunión de orientación, previa a la inspección propiamente dicha, cuando se trata de examinar por primera vez un objeto, para dar a los participantes en la revisión una idea del objeto que van a revisar.

3. Preparación individual: Con suficiente tiempo antes de la realización de la inspección, se debe hacer llegar a cada revisor una copia de la documentación asociada al objeto que se va a revisar, junto con una lista de comprobaciones o "*checklist*" enumerando los posibles defectos que se deben intentar localizar. Una lista de comprobaciones contiene una serie de preguntas, y se supone que al intentar dar respuesta a estas preguntas saldrán a la luz los problemas que puedan existir. Cada revisor debe completar la lista y anotar cualquier tipo de pregunta o defecto detectado.

4. Reunión de inspección: Durante la reunión, el presentador o autor del objeto revisado va guiando al resto a través del mismo para comprobar cada uno de los puntos de la lista de comprobación. Hay varias formas de guiar la reunión:

   - ✓ Punto por punto de la lista de comprobación, revisando todo el producto para cada uno de ellos.
   - ✓ Componente a componente del producto, revisando todos los puntos de la lista de comprobación para cada componente.
   - ✓ Por grupos de puntos dentro de la lista de comprobación, revisando todo el producto para cada grupo.
   - ✓ Cualquier otra solución intermedia

   Los defectos que se detecten durante este proceso se añaden a una lista de acciones pendientes.

Hay que tener en cuenta que el objetivo de una inspección es descubrir defectos, no corregirlos. Por otro lado, los revisores deben restringirse a los hechos y ser constructivos. Es misión del moderador asegurarse de que esto se cumple.

Al final de la reunión de inspección, los participantes valoran los resultados de la inspección y se completa un informe. Al finalizar la reunión se pueden producir las siguientes situaciones:

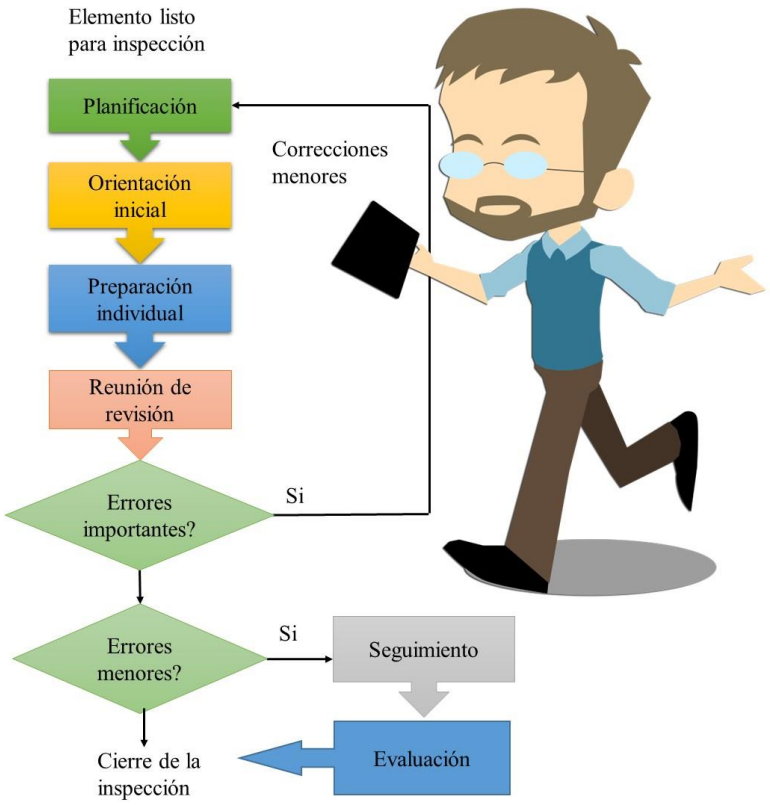
- ✓ Se cierra la inspección sin que se hayan encontrado defectos.
- ✓ Se cerrará la inspección después de que los defectos encontrados se hayan eliminado en la fase de seguimiento.
- ✓ No se cierra la inspección porque se encontraron defectos importantes, y será necesario realizar una nueva inspección.

5. Seguimiento: Durante esta fase, el autor del objeto revisado se encarga de corregir los defectos encontrados y generar un informe en el que se especifican las acciones correctivas realizadas para eliminar los distintos defectos.

6. Evaluación: Es la última fase y en ella se trata de determinar si se han corregido todos los defectos y si han surgido nuevos problemas durante el proceso de corrección. El moderador se encarga de realizar la evaluación y enviar un informe a la dirección una vez finalizada.

En la figura 31, se muestra el proceso para una revisión.

Figura 31: Proceso para una revisión



Documentos generados en una inspección

- Informe resumen de la inspección: Conclusiones breves de la inspección (una página o dos) para la dirección: Qué se ha revisado, Quién lo ha revisado, Cuál fue la conclusión
- Lista de acciones pendientes: Es un informe para los autores del producto revisado explicando qué es lo que está mal y, si puede ser, cómo corregirlo. Necesita ser claro, pero no muy elaborado. Es un documento técnico y transitorio. No debe llegar a la dirección, para no aburrirlos con tantos datos y para que no puedan usar esta información en perjuicio del proceso de inspección.
- Informe de asuntos relacionados: Para registrar problemas que salen a la luz durante la inspección pero no están relacionados directamente con el objeto revisado, para que sean notificados a la persona responsable.
- Informe del proceso de inspección: Cuando algo ha salido mal en el proceso de inspección en sí mismo.
- Informe final: Para informar a la dirección del cierre de la inspección.

Controles dinámicos

Se llama controles dinámicos a aquellos que requieren la ejecución del objeto que se está probando o de un modelo del mismo.

Métodos de caja negra

En este tipo de métodos, el objeto que se desea probar se ve como una caja negra. Esto quiere decir que la elección de los casos de prueba no se va a basar en el conocimiento que se tenga acerca de la estructura del objeto, sino en el conocimiento acerca de la funcionalidad deseada (descripción funcional).

A la prueba de caja negra también se le llama prueba funcional o prueba orientada al diseño.

Una prueba de caja negra exhaustiva requeriría la generación de un caso de prueba por cada combinación posible (válida o no válida) de los valores de entrada, lo cual resulta imposible en la mayor parte de los casos por producirse una explosión combinatoria. Por eso se utilizan diferentes criterios a la hora de restringir el conjunto de casos de prueba.

Los métodos de selección del conjunto de casos de prueba más usuales son:

- Método de clases de equivalencia: Consiste en dividir las posibles entradas al sistema en clases de equivalencia, de tal forma que todos los miembros de una misma clase de equivalencia prueben las mismas propiedades en el sistema, por lo que sólo va a ser necesario seleccionar un elemento de cada clase de equivalencia. El procedimiento para hacer pruebas con esta técnica puede ser visto en el anexo procedimientos del presente documento.
- Análisis de valores frontera o valores límite: Consiste en seleccionar como casos de prueba aquellos valores de entrada que caen en la frontera de las clases de equivalencia (justo a un lado, justo al otro y justo en la frontera).
- Grafos causa/efecto y tablas de decisión: Consiste en crear un grafo causa/efecto a partir de las

especificaciones, y seleccionar suficientes casos de prueba como para asegurar la cobertura del grafo. Se llama causas a las características de los datos de entrada y efectos a las clases de salidas que puede proporcionar el programa. A partir del grafo causa/efecto se construye una tabla de decisión que refleje las dependencias entre causas y efectos. Lo que se hace entonces es reducir la tabla de decisión y seleccionar sólo un caso de prueba para todas las causas que producen el mismo efecto, o para cada columna de la tabla de decisión.

- Adivinación de errores: Consiste en tratar de imaginar cuáles son los errores que se pueden haber cometido con mayor probabilidad, y generar casos de prueba para comprobar dichos errores.

Métodos de caja blanca o caja transparente

En este tipo de métodos, el objeto que se desea probar se ve como una caja blanca. Esto quiere decir que la elección de los casos de prueba se va a basar en el conocimiento que se tenga acerca de la estructura del objeto (diseño detallado, diagramas de flujo de datos y de control, código fuente).

A la prueba de caja blanca también se le llama prueba estructural.

Los métodos de caja blanca se pueden clasificar, a su vez, en dos grupos:

1. Métodos basados en métricas de cobertura

   Todo programa se puede representar mediante un grafo de flujo de control, donde cada nodo es una sentencia o una secuencia de sentencias. Los arcos dirigidos en el grafo representan el flujo de control.

   Para cada conjunto de datos de entrada el programa se ejecutará a través de un camino concreto dentro de este grafo. Cuando el programa incluye estructuras iterativas, el número de posibles caminos en el grafo puede ser infinito.

   Una prueba de caja blanca exhaustiva requeriría la generación de un caso de prueba por cada posible camino. Como esto no es posible, por lo general, se utilizan métricas que dan una indicación de la calidad de un determinado conjunto de casos de prueba en función del grado de cobertura del grafo que consiguen. Las métricas más utilizadas son:

   - ✓ Cobertura de sentencias
   - ✓ Cobertura de segmentos entre decisiones.
   - ✓ Cobertura de decisiones de ramificación
   - ✓ Cobertura de condiciones
   - ✓ Cobertura de todas las combinaciones de condiciones
   - ✓ Cobertura de caminos

2. Métodos basados en métricas de complejidad

   Las métricas de complejidad más utilizadas en la generación de casos de prueba son las de MacCabe:

- ✓ Complejidad ciclomática (arcos - nodos + 2 * número de componentes conexos)
- ✓ Complejidad esencial (complejidad ciclomática - número de subgrafos propios de entrada y salida única)
- ✓ Complejidad real (número de caminos ejecutados)

El procedimiento para hacer pruebas con la técnica de la complejidad ciclomática puede ser visto en el anexo procedimientos del presente documento.

LECTURA ADICIONAL RECOMENDADA

1. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.
2. Pressman, Roger s., Ingeniería de software, un enfoque práctico, sexta edición. Editorial McGraw Hill, México, año 2006.
3. Sommerville, Ian, Ingeniería de software, sétima edición, Editorial Pearson Addison Wesley, España, año 2005.
4. De Antonio, A., Gestión, control y garantía de la calidad del software, Apuntes, Universidad Politécnica de Madrid, año 2004.

EJERCICIOS PROPUESTOS

5.1 Gestión de configuración

- ¿Qué es la Gestión de Configuración del Software (GCS)?
- ¿Por qué el Mantenimiento software ≠ GCS?
- Explique cada una de las actividades de la GCS, dando ejemplos para cada una de ellas
- ¿Qué es la Configuración del Software? ¿Y los Elementos de Configuración?
- De 20 ejemplos para elementos de configuración de software
- ¿Qué es la Línea Base (LB)?
- ¿Qué son las Versiones, Revisiones, Variantes, Configuraciones Alternativas y *Releases*?
- Explique gráficamente las distintas evoluciones de un sistema (Evolución de un sistema, Evolución temporal (revisiones) y Evolución espacial (variantes))
- ¿Qué es un repositorio y que prestaciones da a la GCS?
- Explique gráficamente la evolución de la LB mediante cambios sucesivos
- Explique gráficamente la evolución de la LB mediante cambios simultáneo
- Desarrollo un Plan para la GCS mediante IEEE STD 1042 *Guide to Software Configuration Management* para la empresa VillaSoft de manera que lo pueda aplicar a sus proyectos.

5.2 Control de calidad

i. Cuál es el objetivo de las actividades de Control de Calidad
ii. Dé una breve definición de Defecto, Fallo y Error
iii. Explique que son los controles ESTÁTICOS y DINÁMICOS del software
iv. Explique la diferencia entre las pruebas de caja negra y las pruebas de caja blanca. ¿Alguna de ellas garantiza la ausencia de fallos?
v. Explique la técnica de Caja Blanca del Camino Básico
vi. Aplique la técnica del Camino Básico a los siguientes problemas, generando los respectivos casos de prueba

```
procedimiento: ordenar
      do while queden registros
           leer registro;
          if campo 1 del registro = 0
             then procesar registro;
               guardar en buffer;
               incrementar contador;
             elseif campo 2 del registro = 0
                then reiniciar contador;
                 else procesar registro;
                  guardar en archivo;
           endif
          endif
      enddo
    end
```

```
void burbuja(int n,int *v){
   unsigned long long comp1=0;
        int *p,I,j,temp;
        p=v;
        for(i=0;i<n;i++)
                {
                  for(j=n-1;j>=i+1;j--)
                    {
                      if(p[j]<p[j-1])
                                {
                                  temp=p[j];
                                  p[j]=p[j-1];
                                  p[j-1]=0;
                                   }
                      comp1=comp1+1;
                   }
               }
           }
```

```
function obtener_media : real;

var
n, suma, conta, suma2, total_num : integer;
begin
        read( n );
        repeat
                        if (n >= 20 or n <= 50) then
                                suma := suma + n;
                                conta := conta + 1
                        else
                                suma2 := suma2 + n;
                        total_num := total_num + 1;
                        read (n);
        until n = 0;
        obtener_media := suma / conta;
        write (total_num, suma2);
end;
```

vii. Explique la técnica de Caja Negra de las Particiones Equivalentes

viii. Aplique la técnica las Particiones Equivalentes a los siguientes problemas, se pide:

a. Crear una tabla de clases de equivalencia (las clases deberán ser numeradas) en la que se indiquen las siguientes columnas en cada fila:

- Condición de entrada que se analiza
- Clases válidas y
- Clases no válidas que se generan para la condición
- Regla heurística que se aplica para la generación de las clases de la fila

b. Generar los casos de prueba (especificando la entrada en todos los casos y la salida esperada sólo en los casos válidos) para las clases creadas usando la técnica de particiones de equivalencia, indicando en cada caso las clases que cubre. Enunciar la regla se aplica para derivar los casos a partir de las clases de equivalencia.

Entrada	*Clases Válidas*	*Clases no Válidas*	*Regla*

Casos Válidos (regla, cada caso válido debe cubrir tantas clases válidas como sea posible)

ENTRADA	*SALIDA*

Casos NO Válidos (regla, cada caso no válido debe cubrir una y sólo una clase no válida)

ENTRADA	*SALIDA*

- Un software toma como entrada un archivo cuyo formato de registro es el siguiente:

Numero-empleado	Nombre-empleado	Meses-Trabajo	Directivo

Donde:

- Número-empleado es un campo de números enteros positivos de 3 dígitos (excluido el 000).
- Nombre-empleado es un campo alfanumérico de 10 caracteres.
- Meses-Trabajo es un campo que indica el número de meses que lleva trabajando el empleado; es un entero positivo (incluye el 000) de 3 dígitos.
- Directivo es un campo de un solo carácter que puede ser «+» para indicar que el empleado es un directivo y «-» para indicar que no lo es.
- El programa asigna una prima (que se imprime en un listado) a cada empleado según las normas siguientes:

    - P1 a los directivos con, al menos, 12 meses de antigüedad
    - P2 a los no directivos con, al menos, 12 meses de antigüedad
    - P3 a los directivos sin un mínimo de 12 meses de antigüedad
    - P4 a los no directivos sin un mínimo de 12 meses de antigüedad

- Otro software, donde el usuario puede conectarse al banco por Internet y realizar una serie de operaciones bancarias. Una vez accedido al banco con las consiguientes medidas de seguridad (clave de acceso y demás), la información de entrada del procedimiento que gestiona las operaciones concretas a realizar por el usuario requiere la siguiente entrada:

- Código del banco. En blanco o número de tres dígitos. En este último caso, el primero de los tiene que ser mayor que 1.
- Código de sucursal. Un número de cuatro dígitos. El primero de ellos mayor de 0.
- Número de cuenta. Número de cinco dígitos.
- Clave personal. Valor alfanumérico de cinco posiciones. Este valor se introducirá según la orden que se desee realizar.
- Orden. Puede estar en blanco o ser una de las dos cadenas siguientes:

    - "Talonario"
    - "Movimientos"

  En el primer caso el usuario recibirá un talonario de cheques, mientras que en el segundo recibirá los movimientos del mes en curso. Si este código está en blanco, el usuario recibirá los dos documentos.

- Un software utilizado por una empresa de transporte para calcular la tarifa de cada billete según el trayecto, la antelación en la que se obtiene el billete y la edad del pasajero. Dicha empresa sólo opera viajes entre Santander, Madrid y Barcelona.

  Como datos de entrada toma:

- CiudadOrigen que es un campo que puede tomar los valores "SNT", "MAD" y "BCN".
- CiudadDestino que puede tomar los mismos valores "SNT", "MAD" y "BCN".
- Fecha es un campo del tipo fecha que indica el día en el que se pretende realizar el viaje.

- Edad es un campo numérico positivo de 3 cifras (incluyendo el 000).

  La tarifa obtenida además de estar en función del trayecto realizado, ofrece los siguientes descuentos por antelación y edad del pasajero. Los descuentos no son acumulables y siempre se aplicará el de mayor valor.

  - 15% de descuento sacando el billete con antelación superior a 1 semana y 25% con antelación superior a 1 mes.
  - 30% a los pasajeros con edad inferior a 25 años y 40% a los pasajeros con edad superior a 65 años.

ix. Explique el método de Inspección de Software para el control de la calidad. Use el siguiente diagrama de la figura 31 del documento, para su trabajo.

x. Explique que es una lista de chequeo en el contexto de una Inspección. Investigue algunas listas de chequeo para una Inspección de requisitos, diseño, código, mantenimiento.

TRABAJO DE PROYECTO

Aplicación práctica 8

En estos momentos se encuentra en la etapa de Ejecución de su proyecto, en particular actividad de Ingeniería de Proyecto, con el desarrollo del software definido para el proyecto de la empresa asignada. Sin embargo, como es de su conocimiento, es necesario elaborar un plan de gestión de la configuración ya que es muy probable que deba enfrentar cambios y mantenimiento (plantilla para todos los proyectos de VillaSoft).

Por lo anterior, se requiere la elaboración de un informe según el siguiente formato (sólo la sección de desarrollo, ya que el resto está determinado). Se le recuerda que el plan será genérico pero cada integrante del grupo de trabajo definido, se encargará de "contextualizarlo" para su software en desarrollo, para el cuál recién ha entregado un plan para el mantenimiento del software que se desarrolla en VillaSoft.

III. DESARROLLO

Las secciones de los estándares *IEEE Guide to Software Configuration Management* 1042 y *IEEE Software Configuration Management Plans* 828.

Ayuda:

Para desarrollar el PM, deberá:

i. Analizar qué es un plan de gestión de la configuración (PGC) (apuntes de clase)
ii. Saber cómo se desarrolla un PGC.
iii. Saber la estructura de un PGC según el estándar IEEE *Guide to Software Configuration Management* 1042 y IEEE *Software Conguration Management Plans* 828-1998
iv. Desarrollar el PGC para el proyecto definido, haciendo todas las suposiciones necesarias

CAPÍTULO 6: CERRAR UN PROYECTO

6.1 CIERRE DEL PROYECTO

Desde un punto de vista práctico y formal, el cierre del proyecto oficializa la finalización de todos los compromisos, tanto con la propia organización como con personas externas a ella:

- Certifica y oficializa que se ha cumplido con el alcance y los compromisos con del cliente (para lo cual es necesario que antes del cierre se haya realizado la aceptación). Lo que implica que ya no se hará nada más en relación al proyecto, y que cualquier nueva solicitud será tramitada como un nuevo proyecto.
- Libera totalmente al equipo del proyecto, incluido al gerente del proyecto. Por lo que estos pueden asumir nuevos proyectos.
- Supone el cierre administrativo y financiero de todos los compromisos y derechos adquiridos por el proyecto. Esto incluye el cierre de los contratos con proveedores y cliente, y el cierre financiero del proyecto dentro de la propia organización.

Como cualquier proceso, el cierre del proyecto está formado por un conjunto de pasos que deben realizarse para decir que este se ha completado en su totalidad. Los pasos se indican en la figura 32.

Figura 32: Tareas del proceso de cierre

Tareas Núcleo:

- Aceptación. El primer paso para iniciar el proceso de cierre es asegurarse de que se ha completado la aceptación externa de los entregables, y que esta aceptación se ha formalizado por escrito.
- Cierre del contrato con el cliente (cierre final del proyecto). Una vez recibida la aceptación formal del entregable final se puede proceder a facturar el proyecto, o la parte ligada a la entrega final.
- Cierre de los contratos con proveedores. Recibir la aceptación formal de un entregable implica

que los proveedores que haya participado en su ejecución han completado su trabajo. Por tanto se debe también aceptar su trabajo, liberar los últimos pagos y proceder al cierre de los contratos, de acuerdo a los procesos administrativos existentes en la organización.

- Cierre financiero del proyecto. Una vez realizados los puntos anteriores es necesario asegurarse de que estos han quedado totalmente reflejados en el estado financiero del proyecto, y en el caso de las facturas, que estas se han pagado o cobrado. El proyecto no puede cerrarse oficialmente hasta que todas las facturas han sido pagadas.
- Cierre administrativo del proyecto. Una vez que se ha liberado el equipo y se ha cerrado financieramente el proyecto (todos los ingresos y gastos están imputados y ejecutados), podemos cerrar el proyecto administrativamente. Esto habitualmente consiste en un proceso interno de la organización que debe ser hecho por el director del proyecto. La importancia práctica de esto es el hecho de informar formalmente a la organización sobre la finalización del proyecto, y el cálculo final de los resultados económicos del proyecto.

Tareas Facilitadoras:

- Liberación del equipo interno. De igual forma que con los proveedores, el equipo interno que ha participado en el proyecto queda liberado en el momento de que el entregable final es aceptado. A partir de este punto, cualquier implicación adicional debería ser considerada como un nuevo encargo. Si la organización trabaja con órdenes de trabajo, esta liberación se oficializa con la aprobación y cierre de la orden de trabajo.
- Lecciones aprendidas y documentación del proyecto. Las lecciones aprendidas y la documentación permiten ampliar y actualizar la base de datos de la empresa de cara a la planificación de nuevos proyectos, y suponen la base sobre la que trabajar los procesos de mejora.

LECTURA ADICIONAL RECOMENDADA

5. Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.
6. Pressman, Roger s., Ingeniería de software, un enfoque práctico, sexta edición. Editorial McGraw Hill, México, año 2006.
7. Sommerville, Ian, Ingeniería de software, sétima edición, Editorial Pearson Addison Wesley, España, año 2005.

EJERCICIOS PROPUESTOS

1. Explique la diferencia entre cierre administrativo y cierre del contrato para un proyecto.
2. Busque en la Web un formato para un documento formal para cada uno de los tipos de cierre mencionados.

TRABAJO DE PROYECTO

Aplicación práctica 9

Ha concluido el proyecto, se requiere el cierre formal. Para ello, se debe hacer un cierre administrativo y cierre del contrato para un proyecto. Busque en la Web un formato para un documento formal para cada uno de los tipos de cierre mencionados.

Debe incluir a lo menos:

i. Título del proyecto
ii. Jefe del proyecto
iii. Empresas cliente y ejecutora
iv. Requisitos principales
v. Logros
vi. Firmas

CONCLUSIONES

Considerando que uno de los requerimientos de la formación de ingenieros, en particular los informáticos “es que los estudiantes resuelvan problemas reales en contextos reales”, en el curso denominado “Ingeniería de Software”, se propuso que los proyectos sean la base del aprendizaje por parte de los estudiantes. Un desafío importante para este curso fue enseñar sobre y en base a proyectos y que los estudiantes fueran capaces de plantear proyectos de software.

Este material didáctico apunta a apoyar el proceso de enseñanza-aprendizaje en base a proyectos, Cada capítulo, ejemplos y ejercicios, apuntan en ese sentido.

Es importante agradecer a muchos autores anónimos que fueron consultados en años de docencia que fueron quedando en el olvido, pero sus ideas y aportes, seguro están plasmados en estas líneas.

Se espera una evaluación por parte de los estudiantes de este material didáctico para introducir cambios si son necesarios.

REFERENCIAS BIBLIOGRÁFICAS

Abascal, J., Aedo, I., Cañas, J., Gea, M., Gil, A., Lorés, J., Martínez, A., Ortega, M., Valero, P., Vélez, M., "La interacción persona-ordenador", Editor: Jesús Lorés, ISBN: 84-607-2255-4, 2001.

Alsmadi, I., Abul-Huda, B. *Improving Understandability in Teaching of Software Engineering and Connectivity with the Industry*. IEEE Global Engineering Education Conference (EDUCON) – Learning Environments and Ecosystems in Engineering Education, 22-25, Amman, Jordan, 4-6 April (2010)

Arnowitz, J., et al., Effective Prototyping for Software Makers, Morgan Kaufmann, Elsevier, Inc., 2007.

Beck. K., Extreme Programming Explained. Embrace Change, Pearson Education, 1999. Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

Baecker R. M., Gruden J., Buxton William A. S. y Greenberg S. Readings in human-computer interaction: toward the year 2000. Morgan Kaufmann Publishers, San Mateo, CA, 1995

Bracken, B., *Progressing from student to professional: the importance and challenges of teaching software engineering*. Journal of Computing Sciences in Colleges, 19(2), 358-368 (2003)

Estrada, J. (2015). Análisis de los estándares internacionales más utilizados en la gestión de proyectos. En J. Estrada, Análisis de los estándares internacionales más utilizados en la gestión de proyectos. Buenos Aires: UP.

Guarasa, J., Montero, J.M., San-Segundo, R., Araujo,A., Nieto-Taladriz, O. *A Project-Based Learning Approach to Design Electronic Systems Curricula*, IEEE Trans. on Education, 49(3), 389-397 (2006)

Herrera, V., "Desarrollo de un Plan de Gestión de Mantenimiento de Software para el Departamento de Sistemas de la Universidad Politécnica Salesiana Basado en la norma ISO/IEC 14764:2006, Universidad Politécnica Salesiana, Tesis de pregrado, Cuenca, Ecuador, 2015.

IEEE, IEEE Standards Collection: Software Engineering, IEE Standard 620.12-1990, IEEE, 1993.

Kolmos, A., De Graaff, E., Du, X. *Diversity of PBL-PBL learning principles and models.* Research on PBL practice in engineering education, Sense Publishers, 9-21, Rotterdam, Netherlands, (2009)

Leiva, I. (2013). Aplicación de Metodologías Ágiles para el desarrollo de Software en Dispositivos Móviles. Tesis Magister en Ingeniería de Software. Universidad de Tarapacá, Arica, Chile. 2013.

Liu, J.; Marsaglia, J., Olson, D., *Teaching software engineering to make students ready to real world*. Journal of Computing Sciences in Colleges, 17(6), 43-50 (2002)

Myers B. A. Y Rosson M. B. «Survey on user interface programming» en CHI'92 Conference Proceedings on Human Factors in Computing Systems (BAUERSFELD P., BENNETT J. y LYNCH G., eds.), pág. 195-202. ACM Press, Nueva York, NY, 1992.

Negroponte N. The architecture machine. Towards a more human envi-ronment. The MIT Press, 1970

Negroponte N. Being Digital. Vintage books, Nueva York, NY, 1994.

Norman D. The invisible computer. The MIT Press, 1998.

Martin, J., Rapid Application Development, MacMillan Publishing Co. ed. 1990.

Pressman, Roger S., Ingeniería de software, un enfoque práctico, sexta edición. Editorial McGraw Hill, México, año 2006

Project Management Institute. (2013). Guía de los Fundamentos Para la Dirección de Proyectos (Guía del PMBOK®)-Quinta Edición (SPANISH). Project Management Institute.

Reeves, T., Herrington, J., Oliver, R., *Authentic activities and online learning*. Proceedings 25th HERDSA Annual Conference, 562-562, Perth, Western Australia, 7-10 July (2002).

Savin-Baden, M., *Challenging Models and Perspectives of Problem-Based Learning*. Management of Change; Implementation of Problem-Based and Project-Based Learning in Engineering. Sense Publishers, 9-29, Rotterdam, Netherlands, (2007).

Savin-Baden, M., *Problem-Based Learning in Higher Education: Untold Stories: Untold Stories*. McGraw-Hill Education, London, UK (2000).

Sommerville, Ian, Ingeniería de software, sétima edición, Editorial Pearson Addison Wesley, España, año 2005.

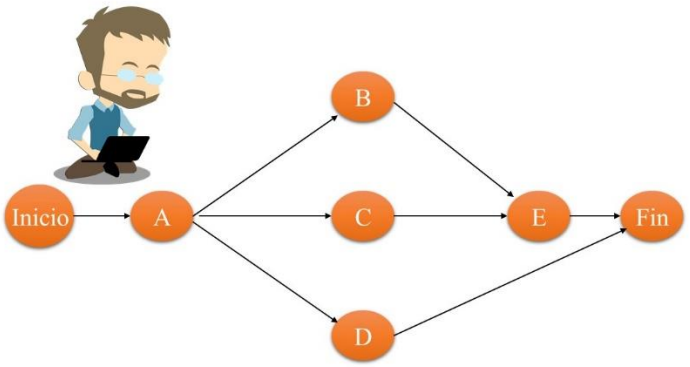
Stiller, E., Lebland, C., *Effective software engineering pedagogy*, Journal of Computing Sciences in Colleges, 18(2), 124-134 (2002).

ANEXOS - PROCEDIMIENTOS

MÉTODO DE LA RUTA CRÍTICA

La Ruta Crítica es la ruta más larga a través de la red. Determina la longitud del proyecto. Toda red tiene al menos una ruta crítica. Es posible que haya proyectos con más de una ruta crítica.

Para el ejemplo de las tareas para construir una casa, ¿cuál es la ruta crítica?. Para ello, primero se construye una red como en la siguiente figura.



Este proyecto tiene tres rutas posibles:

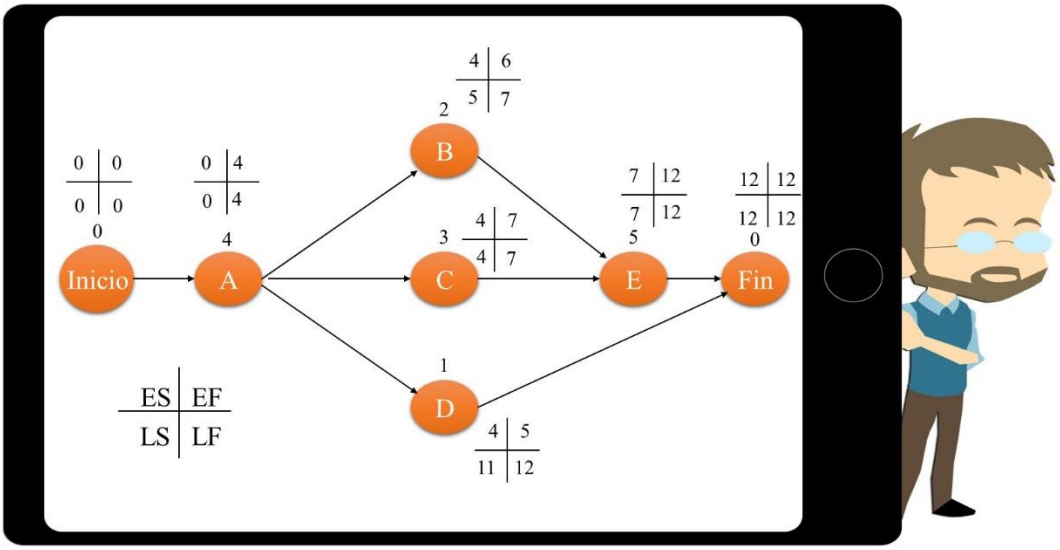
- Inicio – A – B – E – Fin
- Inicio – A – C – E – Fin
- Inicio – A – D – Fin

¿Cuál es la duración de cada una?, ¿Cómo se encuentra la ruta crítica? Son dos preguntas importantes para un gestor de proyecto. Para esto es necesario:

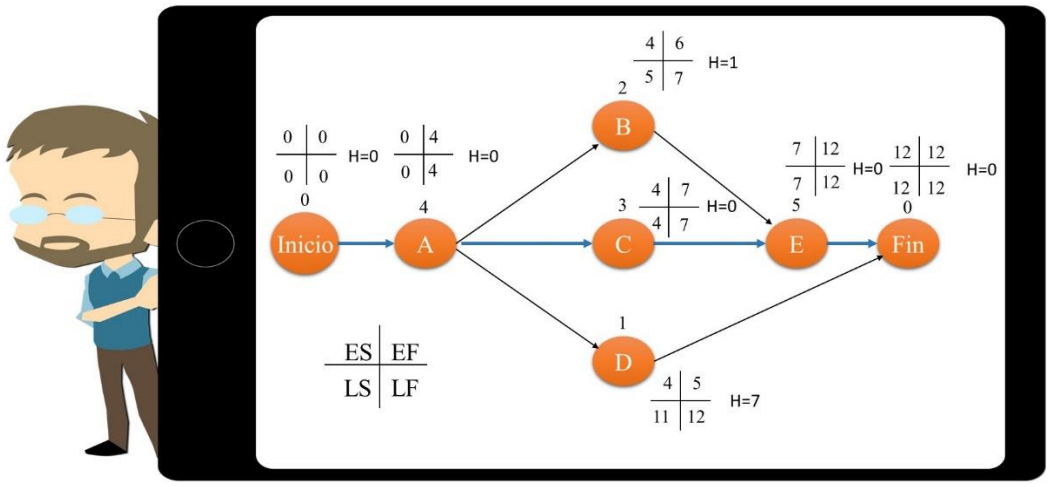
- Agregar a la red los tiempos de cada actividad. Los tiempos se agregarán en cada nodo. Las flechas sólo representan la secuencia de las actividades.
- Para cada actividad se calcularán 4 tiempos. Se denotarán ES, EF, LS, LF:

  - ✓ Tiempo de inicio temprano: Es el tiempo más temprano posible para iniciar una actividad. ES = EF más alto de la(s) actividad(es) anterior(es).
  - ✓ Tiempo de terminación temprano: Es el tiempo de inicio temprano más el tiempo para completar la actividad. EF = ES de la actividad más duración de la actividad. El ES y el EF se calculan recorriendo la red de izquierda a derecha.
  - ✓ Tiempo de terminación más lejana: Es el tiempo más tardío en que se puede completar la actividad sin afectar la duración total del proyecto. LF = LS más bajo de la(s) actividad(es) próxima(s).
  - ✓ Tiempo de inicio más lejano: Es el tiempo de terminación más lejano de la actividad anterior menos la duración de la actividad. LS = LF de la actividad – duración de la actividad. Para calcular LF y LS la red se recorre de derecha a izquierda

  Estos tiempos se ven reflejados en la figura siguiente:



- Después de calculados los cuatro tiempos de cada actividad, se calculan las holguras. La holgura es el tiempo que se puede atrasar una actividad sin afectar la duración total del proyecto. H = LF – EF.
- La ruta crítica se encuentra como aquella ruta para la cual todas sus actividades tienen holgura igual a cero. Generalmente se marca en la red la ruta crítica. En este caso es la ruta: Inicio – A – C – E – Fin, como se muestra en la siguiente.



MÉTODO DE KARNER

Pasos a seguir

UCP=UUCP∗TCF∗ECF∗PF

Donde:

- UCP = Puntos de casos de uso.
- UUCP = Puntos de casos de uso sin ajustar.
- TCF = Factor de complejidad técnica.
- ECF= Factor de complejidad de entorno.
- PF = Factor de productividad.

UUCP

Peso Actores (AUW) + Peso Casos de Uso (UUCW)

Peso de los Actores sin ajuste (uaw)

En la tabla, se describen los pesos para los distintos tipos de actores y sus respectivos pesos.

Actor	*Descripción*	*Factor*
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una API.	1
Medio	Otro sistema interactuando a través de un protocolo (ej. TCP/IP) o una persona interactuando a través de una interfaz en modo texto.	2
Complejo	Una persona interactuando a través de una GUI o desde una página Web.	3

Pesos de los Casos de Uso (uucw)

Este punto funciona muy similar al anterior, pero para determinar el nivel de complejidad se puede realizar mediante dos métodos: basado en transacciones o basado en clases de análisis. Las transacciones son un conjunto de actividades, las cuales deben ser ejecutadas como un todo, o caso contrario ninguna de ellas. Es decir, las transacciones son un grupo de actividades que se ejecutan de forma completa (éxito) o bien se vuelve al estado previo a la ejecución de la transacción (fracaso), quedando siempre el sistema en un estado consistente. El conteo del número de transacciones pueden ser realizadas por medio del conteo de los pasos (*steps*) que tienen los casos de uso. Pero estos pasos son equivalentes a un viaje de ida y vuelta. En la siguiente tabla se muestra la clasificación y el factor de las transacciones para los casos de uso.

Tipo de Caso de Uso	*Descripción*	*Factor*
Simple	3 transacciones o menos	5
Medio	4 – 7 transacciones	10
Complejo	Más de 7 transacciones	15

Ajuste de los casos de uso

Se trata de la cuantificación de características no funcionales del sistema: se toman en cuenta factores de técnicos y factores ambientales, cada factor se evalúa de acuerdo a las tablas siguientes, respectivamente.

Descripción	*Valor*
Irrelevante	De 0 a 2
Medio	De 3 a 4
Esencial	5

Descripción	*Valor*
Sin experiencia, motivación, estabilidad, personal	De 0 a 2
Promedio	3
Amplia experiencia, motivación, estabilidad, personal	De 3 a 5

Factores de complejidad técnicos

En la siguiente tabla, se describen los factores de complejidad técnicos.

Factor	*Descripción*	*Peso*
T1	Sistema distribuido.	2
T2	Objetivos de performance o tiempos de respuesta.	1
T3	Eficiencia del usuario final.	1
T4	Procesamiento interno complejo.	1
T5	El código debe ser reutilizable.	1
T6	Facilidad de instalación.	0,5
T7	Facilidad de uso.	0,5
T8	Portabilidad.	2
T9	Facilidad de cambio.	1
T10	Concurrencia.	1
T11	Incluye objetivos especiales de seguridad.	1
T12	Provee acceso directo a terceras partes.	1
T13	Se requiere facilidades especiales de entrenamiento a usuario.	1

Para obtener el factor técnico final, se debe multiplicar cada ítem por el grado de influencia sobre el sistema y así se obtiene la suma llamada Tfactor, obteniendo la siguiente fórmula:

$$TFactor = \sum (Valor * Peso)$$
$$TCF = 0.6 + (0.01 * TFactor)$$

Factores de complejidad ambientales

En la siguiente tabla, se describen los factores de complejidad ambientales.

Factor	*Descripción*	*Peso*
E1	Familiaridad con el modelo del proyecto utilizado.	1.5
E2	Experiencia en la aplicación.	0.5
E3	Experiencia en orientación a objetos.	1
E4	Capacidad del analista líder.	0.5
E5	Motivación.	1
E6	Estabilidad de los requerimientos.	2
E7	Personal part – time.	-1
E8	Dificultad del lenguaje de programación.	-1

Para obtener el factor ambiental final, se debe multiplicar cada ítem por el grado de influencia del sistema, y así se obtiene la suma llamada Efactor, obteniendo la siguiente fórmula:

$$EFactor = \sum (Valor * Peso)$$
$$EF = 1.4 + (0.03 * EFactor)$$

UCP

Se calcula de la siguiente manera: UUCP * TCF * EF

MÉTODO PARA EL DISEÑO DE METÁFORAS- LENGUAJE VISUAL

Las metáforas pueden conseguir su efectividad a través de la asociación con organizaciones. Se puede asociar estructuras, clases, objetos, atributos a nombres u operaciones. Se puede asociar procesos, algoritmos a verbos.

Nombres – operaciones

Ejemplos típicos de conceptos de nombres u operaciones son:

- Escritorio: Dibujos, archivos, carpetas, papeles, clips, notas de papel.
- Documentos: Libros, capítulos, marcadores, figuras; periódicos, secciones; revistas, artículos; cartas; formularios.
- Fotografía: Álbums, fotos, portafotos.
- Televisión: Programas, canales, redes, anuncios comerciales, guías.
- Grabaciones: Disco compacto, casete, pistas.
- Pila de cartas: Cartas, pilas.
- Juegos: reglas del juego, piezas del juego, tablero de juego.
- Películas: Rollos, presentaciones, películas, teatros.
- Contenedores: Estanterías, cajas, compartimentos.
- Árboles: Raíces, tronco, ramas, hojas.
- Red, diagrama, mapa: nodos, enlaces, regiones, etiquetas, leyenda.
- Ciudades: Regiones, casas, habitaciones, ventanas, mesas.

Algoritmos - verbos

Ejemplos típicos de conceptos de acción y las relaciones con los objetos son:

- Mover: navegar, conducir, volar
- Localizar: apuntar, tocar, enmarcar elemento(s)
- Seleccionar: tocar elemento, grabar elemento, poner dedo en elemento y moverlo
- Crear: añadir (nuevo), copiar
- Borrar: tirar, destruir, perder, reciclar, borrar (temporal o permanentemente)
- Evaluar: Mover botón, desplazar puntero, rodar, girar
- Vaciar, flujo: agua (tubos, ríos), electricidad

Diseño de un conjunto de metáforas (lenguaje visual)

Los pasos que se siguen son:

1. Escoger los objetos que están implicados
2. Asociar un elemento visual a los objetos anteriores
3. Escoger los verbos asociados a las acciones que se pueden dar
4. Construir un elemento visual para el punto anterior

Ejemplo:

Se requiere diseño de un conjunto de metáforas que permitan cubrir la gestión de una librería de imágenes electrónica (Abascal, et al., 2001).

1. Escoger los objetos que están implicados

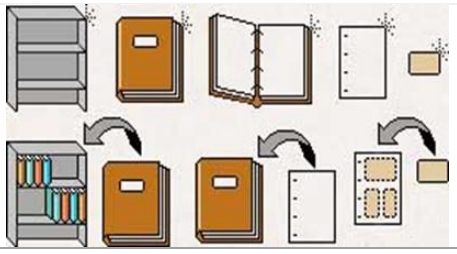
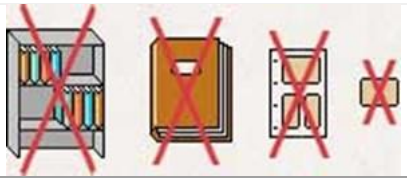
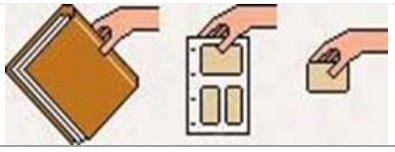
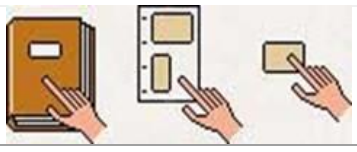
   Estantería, álbum, página, foto

2. Asociar un elemento visual a los objetos anteriores

3. Escoger los verbos asociados a las acciones que se pueden dar

   - Añadir estantería, álbum, página, foto.
   - Borrar estantería, álbum, página, foto.
   - Mover álbum, página, foto.
   - Seleccionar álbum, página, foto.

4. Construir un elemento visual para el punto anterior

Acción	***Metáfora***
Crear	
Borrar	
Mover	
Seleccionar	

MÉTODO PARA ENTREVISTAS

¿Qué es?

Una entrevista es una reunión "cara a cara" entre el o los analista/s y una o más personas: usuarios o implicados en el sistema, gestores, auditores, programadores, etc...

- Da la oportunidad de tratar al usuario y ayuda a disminuir su natural resistencia
- La entrevista puede ser individual o en grupo
- La información se obtiene de manera verbal, ayuda a descubrir errores de entendimiento, falsas expectativas, resistencia hacia el nuevo proyecto...
- Es una forma de conversación, no sólo de interrogación
- Es ideal para recoger opiniones, comentarios, ideas, sugerencias
- Introducir al usuario en los conceptos de modelo subyacente.
- Generar un puente entre el usuario y el desarrollo del software.
- Ayudar al desarrollo del sistema.
- Aprender sobre el dominio.
- Informarse sobre las necesidades y preferencias de los eventuales usuarios del software.
- Y naturalmente, para extraer los requerimientos del usuario.

¿Para qué sirven?

Las entrevistas constituyen el medio de obtener información sobre:

- Requerimientos de usuario
- Funcionamiento del sistema actual
- Organización de la Unidad
- Responsables y funciones de los usuarios
- Etc.

¿Por qué prepararlas?

- Las entrevistas pueden tener un contenido muy denso
- Evitar la posibilidad de ambigüedad
- Evitar malas interpretaciones o falta de comunicación
- Evitar frases del cliente del tipo: “Sé que cree que entendió lo que piensa que dije, pero no estoy seguro de que se da cuenta de que lo que escuchó no es lo que yo quise decir”

Posturas negativas del entrevistado

“Me haces perder demasiado tiempo"

- Puntualidad
- Mantener la entrevista dentro del tema previsto
- Preguntarle al entrevistado si dejaría que un arquitecto le construyera una casa sin hacerle un

conjunto de entrevistas previas de cómo quiere la casa.

“Está en peligro mi puesto de trabajo"
- No soy la persona adecuada para asegurarle lo contrario
- Hacer saber el temor de los entrevistados a su superior

“Tú no conoces el negocio, entonces, ¿Cómo nos puedes decir cómo ha de ser el nuevo sistema?"
- Efectivamente¡ Se realiza las entrevistas para descubrir como necesita el usuario que sea el nuevo sistema

Ventajas v/s Desventajas

En la siguiente tabla se pueden apreciar las ventajas y desventajas de las entrevistas.

Ventajas	*Desventajas*
• Mediante ellas se obtiene una gran cantidad de información correcta a través del usuario. • Pueden ser usadas para obtener un pantallazo del dominio del problema. • Son flexibles. • Permiten combinarse con otras técnicas.	• La información obtenida al principio puede ser redundante o incompleta. • Si el volumen de información que se maneja es alto, requiere mucha organización de parte del analista, así como la habilidad para tratar y comprender el comportamiento de todos los involucrados.

Tipos

En esta técnica existen dos modalidades.

- Las entrevistas no estructuradas

  - ✓ Esta es la técnica más popular utilizada en la obtención de información para la especificación de requisitos del sistema software.
  - ✓ Es una técnica altamente dependiente de las destrezas y conocimientos comunicacionales tanto del entrevistador como del entrevistado.
  - ✓ Es una interacción sistemática en la que no existe estructura ni plan previo de objetivos por lo que se utiliza generalmente en las primeras sesiones a modo de orientación general.
  - ✓ Puede utilizarse con todos los que tienen interés en el desarrollo del producto.
  - ✓ Aunque no existe una preparación fina de la sesión el IR debe delimitar el alcance de la sesión y evitar profundizar en demasía sobre un problema en particular.
  - ✓ Su libertad de formato permite capturar información gran cantidad de información y mucha de ella imprevista, pero consume bastante tiempo.

- Las estructuradas

  - ✓ Las entrevistas estructuradas tienen características definidas, involucran una cuidadosa pre-planificación del cuestionario y su orden, y especificación de cosas.
  - ✓ La idea de que las entrevistas sean planificadas y estructuradas, anticipadamente, es que podrían llevarnos a hacer entrevistas más eficientes.
  - ✓ A pesar de que consumen bastante tiempo de preparación y de que pueden contener

problemas de lenguaje como ambigüedad, son una excelente forma de profundizar en situaciones particulares.
- ✓ En la entrevista, el usuario analiza y comenta cada proposición indicando cualquier acuerdo o alteración de ésta.
- ✓ El resultado de la entrevista estructurada se transcribe en un informe de cambios en la Definición de requisitos y/o especificación de requisitos.

Proceso

Sin embargo, independiente del tipo que se use para realizar la educción de requisitos, se debe tener en cuenta lo siguiente. El proceso para capturar requisitos mediante una entrevista con el cliente y usuario, debería dividirse en 4 fases:

- Identificar los candidatos a ser entrevistados
- Preparar la entrevista
- Entrevistar
- Hacer seguimiento

Actitud del entrevistador

Se debe cuestionar en cada instante:

- ¿Qué me están diciendo? (entender la información)
- ¿Por qué me lo están diciendo? (comprendo el contexto)
- ¿Es completo y consistente lo que me están diciendo? (falta información, es información contradictoria)
- ¿Qué actitud espera de mí el interlocutor? (si yo me veo como un signo de interrogación, el usuario no me considerará de ahora en adelante)

Qué preguntar

- Preguntas sobre el dominio del problema
  - ✓ ¿Qué conocimientos aplica Ud.?
  - ✓ ¿Ud. ocupa técnicas propias o hay otros que lo hacen como Ud.?
  - ✓ ¿Cómo influye el tiempo (y otros factores) en el dominio?

- Preguntas sobre la tarea específica
  - ✓ Jerarquizar y estructurar la tarea
  - ✓ Categorizar los problemas y las soluciones
  - ✓ Definir los datos, sus orígenes y su confiabilidad
  - ✓ ¿Quiénes tienen los problemas y quiénes necesitan las soluciones?
  - ✓ ¿Cómo discrimina el ruido dentro del proceso de solucionar una tarea?

- Preguntas sobre el usuario?
  - ✓ ¿Quién es el usuario?

- ✓ ¿Qué necesita saber el usuario?
- ✓ ¿Qué espera el usuario?

Generalidades

- Revisión de Documentos:

  - ✓ Se tratan de determinar posibles requerimientos sobre la base de inspeccionar la documentación utilizada por la empresa; por ejemplo, boletas, facturas, remitos, etc.
  - ✓ Esta herramienta sirve más que nada como complemento de las demás técnicas, y nos ayuda a obtener información que de otra manera sería sumamente difícil conseguir. Por ejemplo, en las facturas podemos encontrar información que no se pensaba manejar y que en definitiva resulta de suma utilidad, como un número propio de la empresa que se utiliza para saber el orden que tiene la factura en la carpeta y que permite encontrar las copias del documento con mayor rapidez.
  - ✓ En definitiva, se debe recolectar cualquier formulario o documento que sea utilizado para registrar o enviar información.

- Observación:

  - ✓ Observar cómo se hacen las cosas es una buena manera de entender lo que estas requieren.
  - ✓ Conectarse íntimamente con la cultura de la organización, vivirla, es una herramienta que debe ser tomada en cuenta.
  - ✓ Siempre tenemos que estar atentos a lo que sucede en el entorno de la organización; por ejemplo, ver cómo resuelven un problema que surge, como un llamado telefónico que puede ocurrir mientras estamos presentes.
  - ✓ Dentro de la estrategia de observar tenemos que tratar de buscar estructuras y patrones.
  - ✓ La estructura del trabajo para los usuarios suele ser invisible, por lo que será nuestro trabajo realizar las abstracciones necesarias.

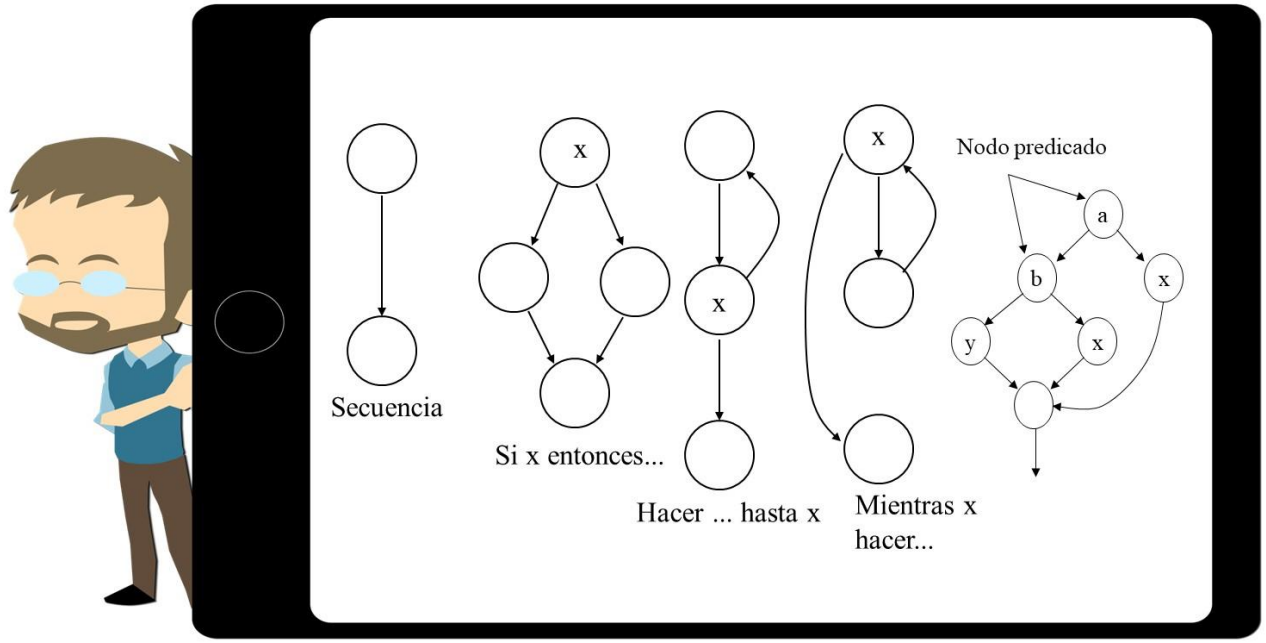
MÉTODOS PARA EL CONTROL DE CALIDAD

Prueba del camino básico

Se trata de obtener una medida de la complejidad lógica de un diseño procedimental y usar esta medida como guía para la definición de un "conjunto básico" de caminos de ejecución.

Notación de grafo de flujo

En la siguiente figura, se presentan los distintos grafos para cada una de las estructuras de control.

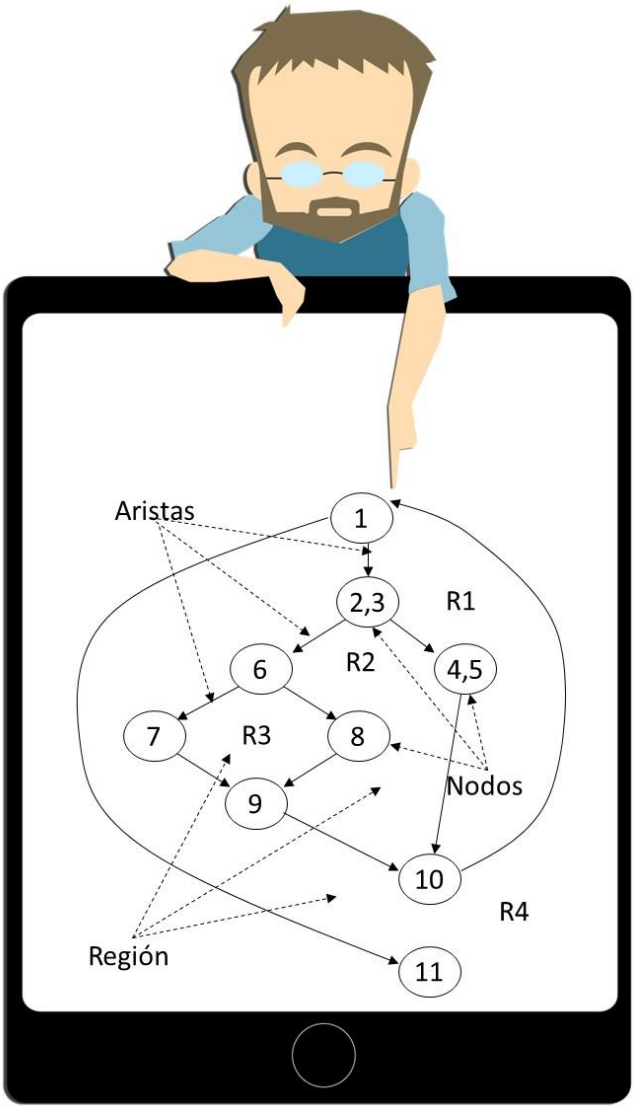


Complejidad Ciclomática

- Es una métrica que proporciona una medición cuantitativa de la complejidad lógica de un programa.
- Camino: secuencia de sentencias encadenadas desde la sentencia inicial del programa hasta su sentencia final.
- En el contexto de una prueba de camino básico, el valor calculado como complejidad ciclomática define el número de "caminos independientes" del conjunto básico de un programa y nos da un límite superior para la cantidad de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez
- Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición
- En términos del grafo de flujo, un camino independientes se debe mover por una arista que no haya sido recorrida anteriormente a la definición del camino

Ejemplo:

El grafo de la figura, caminos independientes serían:

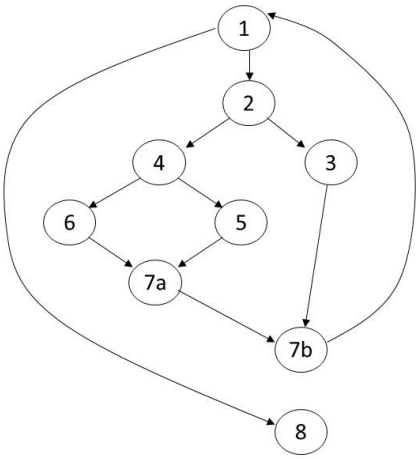


Camino 1: 1-11
Camino 2: 1-2-3-4-5-10-1-11
Camino 3: 1-2-3- 6-8-9-10-1-11
Camino 4: 1-2-3-6-7-9-10-1-11

Un nuevo camino introduce una nueva arista. El camino 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11, no es camino independiente, ya que es una simple combinación de caminos ya especificados y no recorre ninguna nueva arista. Los caminos 1, 2, 3 y 4 componen un conjunto básico para el grafo de flujo de la figura.

Ejemplo: (ordenar) obtenga el grafo de flujo

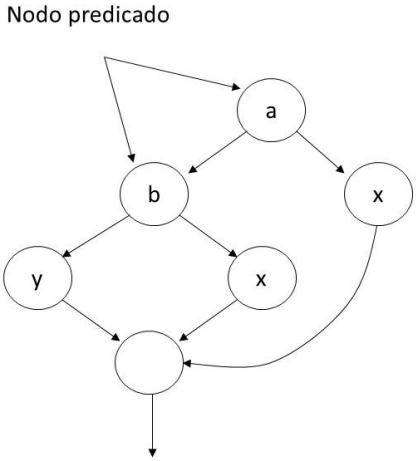
```
    procedimiento: ordenar
1:    do while queden registros
          leer registro;
2:        if campo 1 del registro = 0
3:           then procesar registro;
               guardar en buffer;
               incrementar contador;
4:           elseif campo 2 del registro = 0
5:              then reiniciar contador;
6:              else procesar registro;
                guardar en archivo;
7a:          endif
          endif
7b:   enddo
8  end
```



Condiciones Compuestas:

Se cuenta un nodo aparte por cada una de las condiciones a y b de la sentencia IF a OR b

```
….
IF a OR b
  then procedimiento x
  else procedimiento y
ENDIF
```



Diseño de casos de prueba

Si se diseña pruebas que provoquen la ejecución de los caminos en el conjunto básico, se garantizará que se habrá ejecutado al menos una vez cada sentencia del programa y que cada condición se habrá ejecutado en sus vertientes verdadera y falsa. Este camino básico no es único. Cómo saber cuántos caminos hemos de buscar: El cálculo de la complejidad ciclomática da la respuesta.

¿Número máximo de caminos independientes?.

Complejidad ciclomática de Mc Cabe, V(G):

V(G) = a - n + 2 ("a", nº de arcos del grafo; "n", nº de nodos)
V(G) = r ("r", nº de regiones del grafo)
V(G) = c +1 ("c", nº de nodos de condición)

Ejemplo: Para el grafo calcular la complejidad ciclomática:

V(G) = 4 regiones
V(G) = 11 aristas – 9 nodos + 2 = 4
V(G) = 3 nodos de condición + 1 = 4

Derivación de casos de prueba: (4 pasos)

1. Usando el diseño o el código como base, se dibuja el correspondiente grafo de flujo.
2. Se determina la complejidad ciclomática del grafo de flujo resultante, V(G).
3. Se determina un conjunto básico de hasta V(G) caminos linealmente independientes.
4. Se prepara los casos de prueba que forzarán la ejecución de cada camino del conjunto básico. (a partir de los caminos, analizar el código para ver qué datos de entrada son necesarios, y qué salida se espera)

Particiones o clases de equivalencia

Cada caso de prueba debe cubrir el máximo nº de entradas

Debe tratarse el dominio de valores de entrada dividido en un nº finito de clases de equivalencia. La prueba de un valor representativo de la clase permite suponer "razonablemente" que el resultado obtenido será el mismo que probando cualquier otro valor de la clase

Método de diseño:

1. Identificación de clases de equivalencia.

   1.1 Identificar las condiciones de las entradas del programa
   1.2 A partir de ellas, se identifican las clases de equivalencia: De datos válidos, De datos no válidos
   1.3 Algunas reglas para identificar clases:

   - Por cada rango de valores, se especifica una clase válida y dos no válidas.
   - Si se especifica un nº de valores, se creará una clase válida y dos no válidas.
   - Si se especifica una situación del tipo "debe ser" o booleana, se identifica una clase válida y una no válida.
   - Si se especifica un conjunto de valores admitidos, y el programa trata de forma distinta cada uno de ellos, se crea una clase válida por cada valor, y una no válida.
   - Si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, debe dividirse en clases menores.

2. Creación de los casos de prueba correspondientes.

   2.1 Asignación de un número único a cada clase de equivalencia.
   2.2 Hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba, se tratará de escribir un caso que cubra tantas clases válidas no incorporadas como sea posible.
   2.3 Hasta que todas las clases de equivalencia no válidas hayan sido cubiertas por casos de prueba, escribir un caso para una ÚNICA clase no válida sin cubrir.

GLOSARIO

1. Adaptabilidad. Facilidad con la que un sistema o un componente pueden modificarse para corregir errores, mejorar su rendimiento u otros atributos, o adaptarse a cambios del entorno.
2. Análisis de requisitos. Proceso de estudio de las necesidades del usuario para conseguir una definición de los requisitos del sistema o del software.
3. Ciclo de vida. Periodo de tiempo que comienza con la concepción del producto de software y termina cuando el producto está disponible para su uso. Incluye las fases de concepto, requisitos, diseño, implementación, prueba, instalación, verificación, validación, operación y mantenimiento, y, en ocasiones, retirada.
4. Codificación. Proceso de descripción de un programa de ordenador en un lenguaje de programación.
5. Comité de gestión de configuración (CGC). Grupo de personas responsable de evaluar y aprobar cambios propuestos a elementos de configuración y garantizar la implementación de los cambios.
6. Compatibilidad. Preparación de dos o más componentes o sistemas para llevar a cabo sus funciones mientras comparten el mismo entorno de hardware o software.
7. Componente. Una de las partes que forman un sistema. Un componente puede ser hardware, software, y puede a su vez subdividirse en otros componentes.
8. CPM. (*Critical Path Method*) Método para el control y la optimización de los costos de operación mediante la planificación adecuada de las actividades que componen un proyecto. Fue desarrollado en 1957 en los Estados Unidos por un centro de investigación de operaciones para la firma Dupont y Remington Rand.
9. Crisis del software. Término acuñado en 1968, en la primera conferencia de la NATO sobre desarrollo de software, con el que se identificaron los problemas que surgían en el desarrollo de sistemas de software.
10. Diseño. Proceso de definición de la arquitectura, componentes, interfaces y otras características de un sistema o de un componente.
11. Diseño de arquitectura. Proceso que define una colección de componentes de software y hardware junto con sus interfaces, para definir el marco de desarrollo de un sistema.
12. Diseño detallado. Proceso de definición y ampliación del diseño preliminar de un sistema o de un componente hasta un grado de detalle suficiente para llevar a cabo la implementación.
13. Diseño funcional. Proceso de definición de las relaciones de trabajo entre los componentes de un sistema.
14. Diseño preliminar. Proceso de análisis de las alternativas de diseño y definición de la arquitectura, componentes, interfaces, estimación de tiempo y tamaño de un sistema o de un componente.
15. Disponibilidad. El grado con el que se mide la accesibilidad de un sistema o de un componente cuando es necesario su uso. Suele expresarse en términos de probabilidad.
16. Elemento de configuración. Parte de un desarrollo de software (planes, software, documentación de especificación y diseño, manuales, etc.) tratada como una unidad independiente en el proceso de gestión de configuración.
17. Escalabilidad. Facilidad con la que un sistema o un componente pueden modificarse para aumentar su capacidad funcional o de almacenamiento.
18. Especificación de interfaz. Documento que especifica las características de interfaz de un sistema o de un componente.
19. Especificación de requisitos de software. Documentación de requisitos fundamentales (necesarios, esenciales e indispensables) de funcionalidades, rendimiento, restricciones y atributos del software, y sus interfaces externas.
20. Estimación por analogía. Modelo de estimación de costes y recursos, basado en la comparación con proyectos de ámbitos y características similares, de los que se conocen sus costos reales por haberse terminado.
21. *Extreme Programming*. Es la más popular de las denominadas metodologías ágiles. Surgida a partir de la metodología de trabajo empleada Kent Beck, Wark Cunningham y Martin Fowler en el desarrollo del

proyecto C3 para Chrysler.

22. Flexibilidad. Facilidad con la que un sistema o un componente pueden modificarse para ser empleado con aplicaciones o en entornos distintos para los que fue construido.
23. Gestión de configuración. Disciplina que aplica la dirección y supervisión técnica y administrativa para: identificar y documentar las características funcionales y físicas de un elemento de configuración, controlar cambios, registrar cambios procesados, registrar el estado de la implementación, informar y verificar la conformidad con los requisitos especificados.
24. Gestión de procesos. Dirección, control y coordinación del trabajo realizado para desarrollar o producir un servicio.
25. Implementación. Proceso de transformación de un diseño en componentes de hardware, software o de ambos.
26. Ingeniería del software. Aplicación de procesos sistemáticos y disciplinados para el desarrollo, operación y mantenimiento de software.
27. Interfaz. Componente de hardware o software que conecta dos o más componentes con el propósito de transmitir información entre ellos.
28. Interfaz de usuario. Interfaz que permite la comunicación entre un usuario y un sistema, o los componentes de un sistema.
29. Línea de base. Conjunto de elementos de configuración, formalmente revisados y aprobados (para su uso interno o para entregar al cliente), que constituyen la base para el desarrollo posterior, y que sólo puede modificarse a través de procedimientos de cambio formales.
30. Mantenimiento. Proceso de modificación de un sistema de software o de un componente, después de su puesta en funcionamiento para corregir fallos, mejorar el rendimiento u otros atributos, o adaptarlo a modificaciones del entorno.
31. Mantenimiento correctivo. Modificación de un sistema de software o de un componente, después de su puesta en funcionamiento para corregir fallos.
32. Mantenimiento perfectivo. Modificación de un sistema de software o de un componente, después de su puesta en funcionamiento para mejorar el rendimiento u otros atributos.
33. Manual de diagnóstico. Documento con la información necesaria para ejecutar procedimientos de diagnóstico de un sistema o de un componente. Identifica errores de funcionamiento y establece cómo solucionarlos.
34. Manual de instalación. Documento que contiene la información necesaria para instalar un sistema o un componente, establecer los parámetros iniciales y preparar el sistema o componente para su uso.
35. Manual de operador. Documento que contiene la información necesaria para iniciar y operar con un sistema o con un componente.
36. Manual de programador. Documento que proporciona la información necesaria para desarrollar o modificar el software de un sistema. Ver también: manual de diagnóstico, manual e instalación, manual de operador, manual de soporte, manual de usuario.
37. Manual de soporte. Documento que contiene la información necesaria para mantener operativo un sistema durante su ciclo de vida. Ver también: manual de diagnóstico, manual de instalación, manual de operador, manual de programador, manual de usuario.
38. Manual de usuario. Documento que contiene la información necesaria para obtener de un sistema o de un componente los resultados deseados.
39. Matriz de trazabilidad. Representación gráfica de las relaciones entre dos o más productos del proceso de desarrollo, generalmente identificada, en las intersecciones de líneas verticales y horizontales. Por ejemplo, para representar la relación entre los requisitos y el diseño de un componente del software.
40. Metodologías ágiles. Estrategias de desarrollo de software que promueven prácticas que son adaptativas en vez de predictivas; centradas en las personas o los equipos, iterativas, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa de cliente.
41. Modelo de ciclo de vida. Representación del ciclo de vida del software.
42. Moore (Ley de). Gordon Moore, co-fundador de Intel afirmó en una entrevista a la revista Electronics, que el número de transistores por pulgada, implementados en los circuitos integrados se duplicaría cada

año. Algo más tarde rectificó este plazo a 18 meses. Desde entonces hasta la fecha se viene cumpliendo esta progresión de crecimiento exponencial.

43. Nivel de integridad: Grado de daño que puede producir un fallo en un sistema. El estándar IEEE 1012-1998 define cuatro niveles de integridad para sistemas de software siendo el grado 1 el propio de sistemas cuyo fallo produce daños de escasa relevancia, y el 4 el que implica pérdidas de vida o graves pérdidas económicas o sociales.
44. Obtención. (Aplicado a requisitos). Proceso en el que se implican las partes cliente y desarrolladora para descubrir, revisar, articular y comprender las necesidades y limitaciones que el sistema debe ofrecer a los usuarios.
45. OO. (Orientación por Objetos) Enfoque para el desarrollo de sistemas de software que representa el dominio de aplicación de forma natural y directa basándose en los objetos que se implican en dicho dominio. Emplea diversos métodos para representar de forma abstracta los objetos, definiendo su estructura, comportamiento, agrupaciones, estados, etc.
46. OOA (Object-Oriented Análisis) Análisis orientado por objetos. Método de análisis que examina los requisitos desde la perspectiva de clases y objetos encontrados en el vocabulario del dominio del problema. v. OO.
47. OOP (Object-Oriented Programming) Programación orientada por objetos. Método de implementación de los programas que los organiza como grupos cooperativos de objetos, cada uno de los cuales representa instancias de una clase, que a su vez forman parte de una jerarquía a través de relaciones de herencia.
48. PERT. (*Program Evaluation and Review Technique*) Método para el control de los tiempos de ejecución de diversas actividades integrantes de proyectos. Fue desarrollado en 1957 por la armada de los Estados Unidos.
49. PERT/CPM. Método para el control de la ejecución de proyectos. Combina principios de los métodos PERT y CPM. Su desarrollo en un proyecto resulta útil para: conocer la probabilidad de cumplimiento de fechas, identificar las actividades con mayor potencial para retrasar el proyecto y evaluar las consecuencias de una desviación.
50. Plan de proyecto. Documento que describe el enfoque técnico y de gestión que seguirá un proyecto. Generalmente, el plan describe el trabajo a realizar, los recursos necesarios, los métodos a utilizar, los procesos a seguir, los programas a cumplir y la forma en la que se organiza el proyecto.
51. Proceso propio. Proceso definido en el modelo de ingeniería, y que junto con el resto de procesos del modelo constituye un valor activo de la organización (Know-how).
52. Producto de software. Conjunto de programas, procedimiento y opcionalmente documentación asociada que se entrega al usuario como resultado.
53. Programa principal. Componente de software, llamado desde un sistema operativo y que a su vez suele llamar a otros componentes de software.
54. Prototipado. Técnica de desarrollo consistente en la construcción de una versión preliminar de parte o de todo un sistema, para evaluar su viabilidad, funcionalidad, tiempos de respuesta, etc.
55. Prototipo. Versión preliminar de un sistema que sirve de modelo para fases posteriores.
56. Prueba de interfaz. Prueba cuya finalidad es evaluar el correcto intercambio de información y control entre componentes.
57. Prueba de sistema. Prueba cuya finalidad es evaluar el grado de conformidad con los requisitos de un sistema completo.
58. Prueba estructural. Prueba que centra su atención en la mecánica interna de un sistema o componente.
59. Prueba formal. Prueba ejecutada según planes y procedimientos de prueba revisados y aprobados por el cliente, usuario o personal de gestión.
60. Prueba funcional. Prueba que ignora la mecánica interna de un sistema o un componente y centra la atención sólo en las salidas generadas como respuesta a determinadas entradas y condiciones de ejecución.
61. Prueba informal. Prueba ejecutada según planes y procedimientos que no han sido revisados y aprobados por el cliente, usuario o personal de gestión.

62. *Rational Unified Process* (RUP). Proceso de Ingeniería del Software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades en las organizaciones de desarrollo de software. Se trata de un proceso integrado en un producto, desarrollado y mantenido por Racional Software, e integrado en su conjunto de herramientas de desarrollo. Se encuentra disponible a través de IBM.
63. Redundancia. Presencia de componentes auxiliares en un sistema para realizar funciones idénticas o similares a las de los componentes principales.
64. Redundancia activa. Uso de elementos redundantes en operación simultánea para prevenir fallos.
65. Redundancia pasiva. Uso de elementos redundantes que permanecen detenidos hasta que ocurre un fallo en el elemento principal.
66. Requisito. Condición o facultad que necesita un usuario para resolver un problema.
67. Requisito de diseño. Requisito que especifica o impone condiciones al diseño de un sistema o de un componente.
68. Requisito de implementación. Requisito que condiciona la codificación o la construcción de un sistema o de un componente.
69. Requisito de interfaz. Requisito que especifica un elemento externo con el que un sistema o un componente deben interactuar; o que establece condiciones, formatos, tiempos u otros factores que deben respetarse en dicha interactuación.
70. Requisito de rendimiento. Requisito que impone condiciones sobre un requisito funcional. Por ejemplo los requisitos que especifican velocidad, precisión o uso de memoria.
71. Requisito físico. Requisito que especifica las características físicas que debe presentar un sistema o un componente de un sistema; por ejemplo, material, longitud o peso.
72. Requisito funcional. Requisito que especifica una funcionalidad que debe realizar un sistema o un componente.
73. Robustez. El grado de capacidad que presenta un sistema o un componente para funcionar correctamente frente a entradas de información erróneas, o carga de trabajo elevada.
74. Scrum. Metodología ágil, aplicada originalmente por Jeff Sutherland y elaborada más formalmente por Ken Schwaber. Scrum aplica principios de control industrial, junto con experiencias metodológicas de Microsoft, Borland y Hewlet Packard.
75. SEI. (*Software Engineering Institute*) Fundación federal norteamericana para la investigación y desarrollo, cofinanciada por el Departamento de Defensa de los Estados Unidos y dependiente de la Universidad Carnegie Mellon.
76. Sistema. Conjunto de procesos, hardware, software, instalaciones y personas necesarios para realizar un trabajo o cumplir un objetivo.
77. Sistema de software. Conjunto de programas de ordenador, procedimientos y opcionalmente la documentación y datos asociados, necesarios para el funcionamiento de un sistema.
78. Sistema intensivo de software. Sistema en el que el principal componente es el software.
79. SLIM. (*Software Lifecycle Management*) Metodologías para estimaciones de duración, costos, control de proyectos y gestión de métricas. Desarrolladas por la comercial QSM
80. Software. Los programas de ordenador, procedimientos, y opcionalmente la documentación y los datos asociados que forman parte de un sistema.
81. Software de sistema. Software diseñado para facilitar o permitir la operación y el mantenimiento de un sistema informático; por ejemplo los sistemas operativos.
82. Software de soporte. Software de ayuda para el desarrollo o mantenimiento de otro software; por ejemplo compiladores, editores y otras utilidades.
83. SQA. (*Software Quality Assurance*) Se aplica a los procesos o a las funciones encaminadas a garantizar que la organización realiza el trabajo de desarrollo, operación o mantenimiento de software conforme a los procedimientos y métodos establecidos para el proyecto.
84. Subsistema. Sistema subordinado a otro mayor.
85. SWEBOK. Siglas de: "*Software Engineering Body Of Knowledge*", proyecto que tiene como finalidad definir y acotar las áreas de conocimiento que comprenden la Ingeniería del Software. En su desarrollo participan: IEEE, ISO/IEC/JTC1/SC, los principales autores de obras de Ingeniería del software: Steve

Mc Connell, Roger Presuman e Ian Sommerville; así como importantes empresas: Racional, SAP, Boeing, Construx, MITRE, Raytheon.

86. Trazabilidad. Grado de relación entre dos o más productos del proceso de desarrollo, especialmente productos que tienen una relación de predecesor – sucesor o de superior – subordinado con otro.
87. Trazabilidad de requisitos. Evidencia de una asociación entre un requisito y sus requisitos origen, su implementación y verificación.
88. Tolerancia a errores. Preparación de un sistema o de un componente para continuar su estado normal de operación, a pesar de la presencia de entradas erróneas.
89. Tolerancia a fallos. Preparación de un sistema o de un componente para continuar su estado normal de operación, a pesar de la aparición de errores de hardware o de software.
90. Validación. Confirmación mediante examen y aportación de pruebas objetivas de que se cumplen los requisitos concretos para un uso determinado.
91. Verificación. Confirmación mediante examen y aportación de pruebas objetivas de que se cumplen los requisitos específicos.
92. Verificación y validación. Proceso que determina si los requisitos de un sistema o de un componente son completos y correctos, si los productos de cada fase cumplen los requisitos o condiciones marcados al inicio de la fase y si el sistema o componente final cumple con los requisitos especificados.