1.1. Ingeniería del Software

Un sistema interactivo se compone básicamente de una parte física (hardware) y de un complejo conjunto de instrucciones (software) que gobiernan funcionalmente las posibilidades que aquella ofrece.

Lo que se conoce como Ciclo de Vida del Software describe el proceso completo del desarrollo del software relativo a un determinado sistema, desde su idea inicial hasta que éste es retirado de circulación.

Esta descripción se establece coordinando una serie de actividades de desarrollo, que son recogidas en una disciplina de la ingeniería conocida como Ingeniería del Software (IS), sobre las que se han basado y siguen basándose los desarrollos actuales de los sistemas computacionales.

1.1.1. Definiciones de Ingeniería del Software

El término de Ingeniería del Software ha sido definido por varios autores y, al igual que hace R. S. PRESSMAN [PRE98], citamos como una de las primeras la propuesta por F. BAUER (1969):

> *La ingeniería del software es el establecimiento y uso de principios robustos de ingeniería con el fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales.*

Esta definición no hace referencia alguna a aspectos como la satisfacción del cliente o de la importancia de realizar mediciones ni de la calidad del producto final.

Posteriormente aparecen nuevas definiciones, siendo hoy en día la más comúnmente aceptada la establecida en IEEE Std 610.12-1990 [IEE93]:

> *Ingeniería de Software: (1) la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software. (2) El estudio de enfoques como en (1).*

La Ingeniería del Software, por tanto, no sólo cubre los aspectos puramente tecnológicos de la producción del citado software, sino que conlleva además la gestión de los presupuestos, de proyectos y de los equipos de desarrollo, así como también de la planificación de dichos proyectos.

Comprende, además, un conjunto de tres elementos clave: Los *modelos de proceso* o métodos, las *herramientas* y los *procedimientos*.

Los primeros indican cómo proceder para construir técnicamente software, las herramientas proporcionan el soporte automático o semiautomático para que los ingenieros software puedan desarrollar los métodos, y los procedimientos definen las secuencias de aplicación de los métodos, siendo los que sirven de enlace entre los métodos y las herramientas.

Para cada uno de los tres elementos citados existen varias propuestas que aunque son diferentes pueden combinarse para una mejor adaptación al proyecto o al equipo de

desarrollo que lo lleve a cabo.

Algunos autores han dedicado gran parte de su trabajo en recopilar y actualizar toda cuanta información al respecto, publicando varias versiones de libros de Ingeniería del Software que hoy en día son considerados como "clásicos". Los más conocidos son, y sin intención de desmerecer a los demás, R.S. PRESSMAN en [PRE00a] y I. SOMMERVILLE [SOM89][SOM00], contando cada uno de ellos con ediciones constantemente revisadas, actualizadas y reeditadas.

1.1.2. Modelos de Proceso de la Ingeniería del Software

Se entiende por proceso un conjunto organizado de actividades que transforma entradas (*inputs*) en salidas (*outputs*). Las descripciones de un proceso juntan o encapsulan conocimientos que podrán reutilizarse.

Podemos considerar ejemplos de procesos básicos en la vida cotidiana el manual de instrucciones del lavavajillas, un libro de recetas de cocina, el manual o libro de estilo para los empleados de un banco o el manual de calidad para la fabricación de medicamentos.

Y por proceso software al conjunto estructurado de actividades requerido para desarrollar un sistema software (especificación, diseño, desarrollo y validación).

Así pues, un Modelo de Proceso Software consistirá en la *representación abstracta del proceso*, constituyendo una descripción de un proceso software desde una perspectiva particular.

Como se ha mencionado anteriormente, existen varios modelos de proceso que describen cómo proceder eficientemente para la producción de software. A pesar de que no es objeto de este trabajo detallar todos y cada uno de ellos[1], creemos conveniente explicar la idea de algunos de ellos, puesto que por su importancia y/o aceptación han marcado la evolución del desarrollo del software y, sobre todo, porque son necesarios para comprender el modelo de proceso que posteriormente se presenta en este trabajo de investigación.

1.1.2.1. Modelo lineal o modelo en cascada

Las dos figuras siguientes ilustran dos variantes del ciclo de vida más conocido para la Ingeniería del Software, denominado según los autores como "modelo lineal" o "modelo en cascada" (*waterfall model*), que sugiere un enfoque sistemático y secuencial del desarrollo del software que comienza a nivel del sistema y progresa a través del análisis, el diseño, la codificación, la prueba y el mantenimiento.

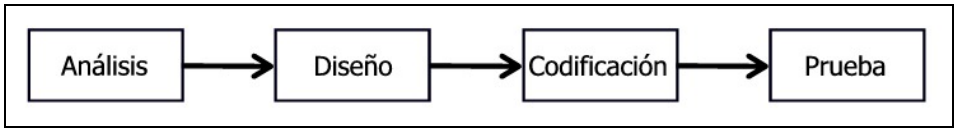

Figura c2_6: Versión 1 del modelo lineal

[1] Es muy fácil además encontrar información detallada de todos los métodos en publicaciones especializadas como los libros anteriormente mencionados.

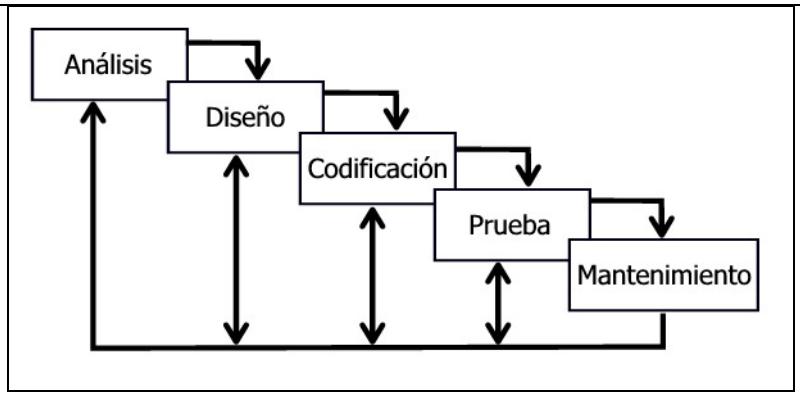

Figura c2_7: Versión 2 del modelo lineal

De los dos esquemas anteriores, más que la misma concepción de una idea, el segundo constituye una versión más evolucionada y realista del primero, proporcionando una realimentación necesaria en todo proceso software.

Las diferentes fases que abarca son[2]:

Ingeniería y análisis del sistema. Debido a que un sistema software siempre forma parte de un sistema mayor, se comienza estableciendo los requerimientos de todos los elementos del sistema y asignando luego algún subconjunto de estos requerimientos al software. Esta visión del sistema es esencial cuando el software debe interrelacionarse con otros elementos tales como hardware, personas o bases de datos. Esta fase abarca los requerimientos globales a nivel de sistema con una pequeña cantidad de análisis y diseño a nivel superior.

Análisis de los requerimientos del software. El proceso de recogida de los requerimientos se centra e intensifica especialmente en el software. Para comprender la naturaleza de los programas que hay que construir, el ingeniero de software o "analista" debe comprender el dominio de la información del software, así como la función, rendimiento e interfaces requeridas. Los requerimientos, tanto del sistema como del software, se documentan y revisan con el cliente.

Diseño. El diseño del software es realmente un proceso multipaso que se enfoca sobre tres atributos distintos del programa: La estructura de los datos, la arquitectura del software y el detalle procedimental. El proceso de diseño traduce los requerimientos en alguna forma de representación del software. Como los requerimientos, el diseño se documenta y forma parte de la configuración del software.

Codificación (o implementación). El diseño debe traducirse en una forma inteligible para las máquinas, cuya tarea se ejecuta en esta fase del proceso. Si el diseño se realiza de una manera detallada esta codificación se producirá de manera mecánica, mientras que si el diseño ha sido pobre el proceso de codificación será más laborioso y más propenso a errores. El resultado de esta fase es lo que comúnmente se denomina "código" o genéricamente "programa".

Prueba (o test). Una vez que se ha codificado el software comienza la prueba del mismo.

[2] Estas fases o etapas no son exclusivas de este modelo: son comunes al resto de métodos y, aunque cada una introduce matices propios, la esencia es la misma.

Dicha prueba se enfoca sobre la lógica interna del software, asegurando que todas las sentencias se han probado, y sobre las funciones externas, esto es, realizando pruebas para asegurar que la entrada definida producirá los resultados que realmente se requieren.

Mantenimiento. El software sufrirá indudablemente cambios después de que éste sea entregado al cliente. Los cambios ocurrirán debido a:

- Que se han encontrado errores.
- Que el software debe adaptarse por cambios del entorno externo (un nuevo sistema operativo, periférico, etc.).
- Que el cliente requiere aumentos funcionales o de rendimiento.

El principal inconveniente del modelo lineal radica en la dificultad de encajar adecuadamente los cambios que se producen durante el proceso.

Otros problemas de este modelo son el particionamiento inflexible de las fases en las que se divide el proyecto software, la dificultad de responder a nuevas necesidades del cliente. Este proceso sólo es válido para el caso ideal en el que los requerimientos están exactamente definidos y detallados desde el principio. La versión 2 del modelo, al tratarse de un modelo iterativo, refleja mejor la realidad del desarrollo software.

1.1.2.2. Modelo de desarrollo evolutivo en espiral

Los modelos evolutivos parten del concepto de que el software, al igual que todos los sistemas complejos, evoluciona con el tiempo [GIL88]. Los condicionantes del producto pueden variar durante el proceso, y éste debe responder eficientemente a la nueva situación.

Estos modelos son repetitivos (iterativos) y se caracterizan por la forma en la que permiten a los ingenieros del software desarrollar versiones cada vez más completas del sistema.

Como es de suponer, existen varios esquemas que dan soporte al concepto del modelo evolutivo, siendo, seguramente, el *modelo en Espiral* [BOH88][PRE00a] el más conocido de ellos.

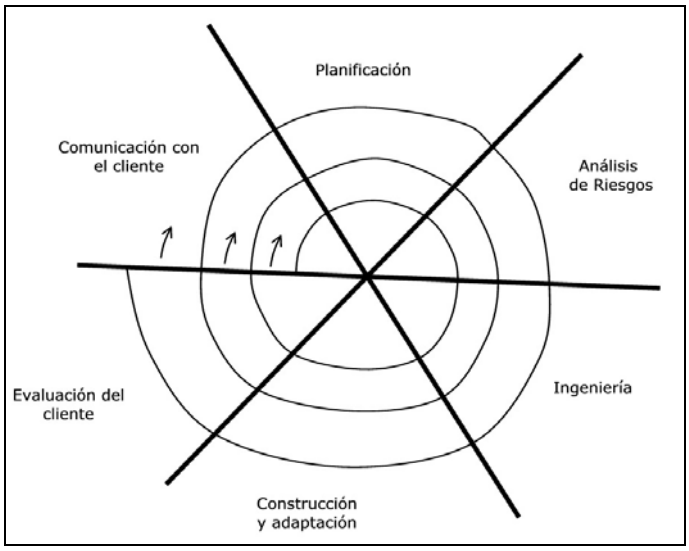

Figura c2_8: Modelo en espiral

Este modelo, que basa su proceso en el *desarrollo rápido de versiones incrementales* del software a implementar, se divide en una serie de regiones de tareas (o actividades de trabajo). Éstas son:

- *Comunicación con el cliente*. Agrupa los aspectos de la comunicación entre el desarrollador y el cliente
- *Planificación*. Definición de los recursos materiales, humanos y temporales del proyecto
- *Análisis de riesgos*. Evaluación y gestión de los posibles riesgos que pueden aparecer durante el desarrollo.
- *Ingeniería*. Construcción de representaciones o prototipos del sistema.
- *Construcción y acción*. Incluye todo lo relacionado con el desarrollo, prueba, instalación y soporte al usuario.
- *Evaluación del cliente*. Obtención de las reacciones del cliente.

Este modelo mantiene el enfoque sistemático de los pasos sugeridos por el ciclo de vida en cascada, pero lo incorpora al marco de trabajo iterativo que refleja de forma más sensata el mundo real [PRE00a].

Los principales problemas de este modelo son la falta de visibilidad del proceso, una pobre estructuración y unos requerimientos técnicos muy específicos, como por ejemplo los lenguajes de prototipado rápido.

1.1.2.3. Modelo de desarrollo evolutivo WinWin

El modelo de desarrollo evolutivo, conocido con el nombre WinWin, que fue propuesto por B. BOHEM [BOH98], es visto como una variación o una evolución del modelo en espiral que, como hemos visto, utiliza una aproximación cíclica para el desarrollo incremental de sistemas software.

WinWin literalmente significa "GanarGanar" y hace referencia a que el desarrollo del proceso software se basa en una *constante negociación entre el cliente y el desarrollador en busca del beneficio mutuo y constante*, o sea, el cliente quiere "ganar" y el desarrollador también quiere "ganar", por lo que el centro de la negociación entre ambos adquiere una especial relevancia en la fase de los requisitos del sistema.

Cada ciclo envuelve cuatro actividades principales:

- Elaboración del sistema o subsistemas y los objetivos, restricciones y alternativas del proceso.
- Evaluar las alternativas respecto a los objetivos y restricciones. Identificar y resolver el mayor número de fuentes de producto y riesgos posibles.
- Elaboración de la definición del producto y del proceso.
- Planificación del próximo ciclo y actualización de la planificación del ciclo de vida. Ello incluye, si es necesario, el particionamiento del sistema en subsistemas a desplegarse en paralelo. Puede incluir, además, la definición de un plan para finalizar el proyecto si éste es de riesgo demasiado alto o no es factible.

Una dificultad proviene de contestar a la pregunta "¿cómo se elaboran los objetivos, las restricciones y las alternativas?". En el modelo WinWin esta tarea se consigue a base de identificar los implicados principales y establecer sus condiciones para "ganar con el sistema". Este concepto es una incorporación importante al modelo en espiral, puesto que si no se

determinan estas condiciones podrá desarrollarse un producto funcionalmente correcto pero totalmente insatisfactorio.

Otro aspecto importante que este modelo añade al modelo en espiral es la introducción de una serie de *hitos tangibles* hacia el proceso software. BOHEM describe tres hitos críticos:

a) Objetivos del Ciclo de Vida (OCV)
b) Arquitectura del Ciclo de Vida (ACV)
c) Capacidad Operacional Inicial (COI)

Los OCV ayudan a describir los objetivos del sistema y conducen a la ACV, sirviendo también para elaborar los objetivos iniciales. Los OCV identifican además los objetivos del sistema y la arquitectura muestra lo lejos que estamos de dichos objetivos.

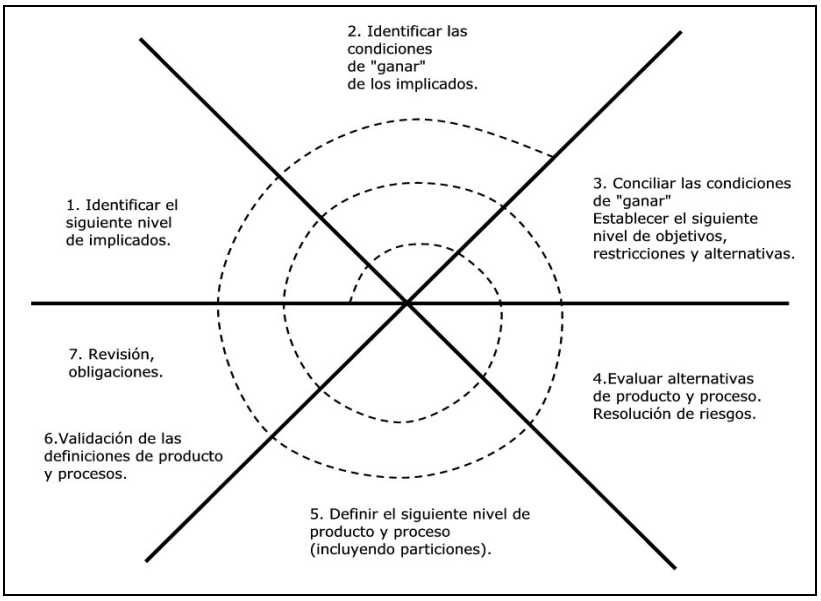

Figura c2_9: Modelo WinWin

1.1.3. La calidad del software

Actualmente los sistemas basados en ordenadores son fundamentales en casi todos los ámbitos de la vida de las personas y su correcto funcionamiento es crucial para el éxito de negocios, de la comunicación entre las personas y para la seguridad de las mismas, por tanto, desarrollar o escoger sistemas software de alto nivel cualitativo constituye un aspecto imprescindible.

La Organización Internacional para la Estandarización ISO definió en el año 1991, entre otros, el estándar ISO/IEC 9126 [ISO91b] que determina los factores que necesita un sistema software para cumplir los parámetros de calidad y el estándar ISO/IEC 14598 [ISO91a] expresando el proceso de evaluación de los mismos.

El concepto de la calidad en sí mismo lo encontramos definido en la ISO/IEC 9126 de la siguiente manera:

> *El conjunto total de características de una entidad (producto, proceso o servicio) que le confieren la capacidad de satisfacer las necesidades establecidas y las necesidades implícitas.*

Por su parte, I. SOMMERVILLE proporciona otra definición que no por ser simple deja de ser precisa:

> *Calidad significa que un producto debe encontrar su especificación* [SOM00].

Las características definidas en ISO/IEC 9126 sirven para las especificaciones de los requerimientos funcionales y también para los no funcionales (calidad interna y externa), tanto desde el punto de vista del cliente como del usuario (calidad en el uso).

La parte primera de dicho estándar [ISO01] define la **calidad interna y externa** de una aplicación basada en los atributos de funcionalidad, fiabilidad, usabilidad, eficiencia, mantenimiento y portabilidad (figura c2_10) y la **calidad en el uso** basada en las características de efectividad, productividad seguridad y satisfacción (figura c2_11).

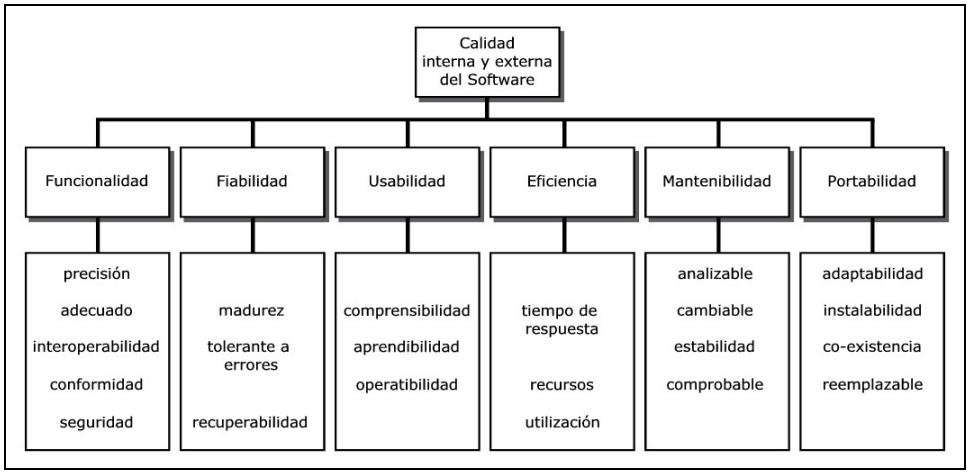

Figura c2_10: Características de la calidad *interna y externa* del software descritas en el estándar ISO/IEC 9126-1

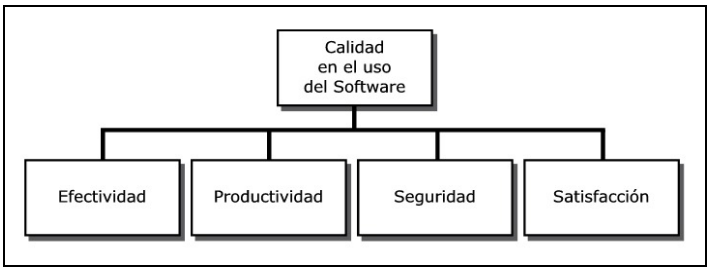

Figura c2_11: Características de la calidad *en el uso* del software definidas en el mismo estándar

El estándar define además un marco de trabajo para el modelo de calidad que explica la relación entre diferentes aproximaciones de calidad en el ciclo de vida del desarrollo software.

1.1.4. Actividades de Protección

Según especifica R.S. PRESSMAN, un proceso software se caracteriza de la siguiente forma

[PRE00a]:

- Se establece un *marco común* que define un conjunto de actividades que son aplicables a todos los proyectos con independencia de su tamaño o complejidad.
- Se define un *conjunto de tareas* que permite adaptar las actividades del marco común de trabajo a las características del proyecto y a los requisitos del equipo de desarrollo.
- Se desarrollan una serie de *actividades de protección* que abarcan todo el modelo de proceso, que son independientes de cualquier actividad y estructura, y existen a lo largo de todo el proceso.

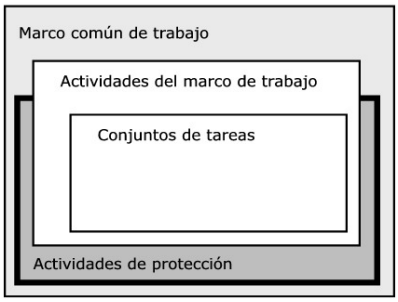

Figura c2_12: Características de todo proceso software

Y las actividades que podemos encontrar en esta última categoría son:

- El seguimiento y control del proyecto.
- Las Revisiones Técnicas Formales (RTFs).
- La Garantía de la Calidad del Software (GCS o *Software Quality Assurance* SQA).
- La Gestión de la Configuración (GC).
- La preparación y producción de documentos.
- La gestión de la reutilización.
- Las mediciones o métricas.
- La Gestión del Riesgo (GR).

Es importante *no confundir estas actividades de protección con las actividades de mantenimiento*, ya que éstas últimas corresponden a una fase del proceso software que empieza una vez el sistema entra en funcionamiento, cuando se ha instalado la versión definitiva del producto. Mientras que las actividades de protección se inician con el inicio del proyecto, continúan una vez instalado y sólo finalizan si el producto deja de dar servicio.

Trataremos a continuación aquellas actividades de protección que tienen relación directa con nuestro trabajo.

1.1.4.1. Gestión de la Configuración (GC)

> *El cambio es un aspecto determinante que inevitablemente se produce durante la vida útil de un producto software.*

Los cambios, como pueden ser la aparición de nuevos sistemas operativos, la necesidad de ofrecer nuevas funcionalidades o cambiar alguna de las existentes, o la aparición de nuevos requisitos, etc., pueden venir ocasionados por una infinidad de factores internos o externos y se pueden producir en cualquier momento y por cualquier razón.

Y, cuando los cambios aparecen puede producirse una situación de confusión (entre los componentes del equipo de desarrollo) provocada en gran medida por no haber analizado el impacto de dicho cambio. La Gestión de la Configuración (GC) es, como anteriormente se ha citado, *una actividad de protección del software que proporciona el mecanismo necesario para identificar, organizar y controlar las modificaciones que sufre el software que implementa un equipo de desarrollo* [BAB86], constituyendo un conjunto de actividades desarrolladas para gestionar dicho cambio a lo largo de toda la vida del software.

La responsabilidad de la Gestión de la Configuración recae en controlar el repositorio completo de todos los aspectos relacionados con el desarrollo del software, incluyendo documentos, modelos de diseño, bases de datos, código, *scripts* de pruebas, acciones concretas y, en general, cualquier aspecto que pueda ser "debido a" o "provocado por" cualquier cambio.

Wayne BABICH, uno de los autores que más han contribuido a la GC, en la introducción de su libro *Software Configuration Management* [BAB86] expone:

> *Cuando entrego el software al cliente no sólo incluyo la versión de la entrega sino que también incluyo la evolución día a día, minuto a minuto del software realizado por el equipo de desarrollo. Evolución controlada significa que usted no solamente entiende lo que tiene cuando lo está entregando, sino que también entiende lo que usted tiene mientras que lo desarrolla. El control ayuda a obtener la máxima productividad minimizando la confusión cuando un grupo de ingenieros de software trabajan junto en un mismo proyecto* [VEN99].

Las actividades de la Gestión de la Configuración son:

- Identificación del cambio.
- Controlar el cambio.
- Garantizar que el cambio se implementa adecuadamente.
- Informar puntualmente del cambio realizado a todos los interesados.

Otra actividad importante de la GC que aparece como consecuencia lógica de las enumeradas anteriormente es la *Organización de* las diferentes *Versiones* existentes de cada elemento software y de su documentación asociada. La evolución del software en sus diferentes versiones suele representarse en forma de grafo validado por una comisión de personas del equipo de desarrollo conocida como Comisión de Control de Cambios.

El resultado de todo este conjunto organizado de actividades queda permanentemente reflejado en los informes conocidos como *Informes del Estado de la Configuración* que permiten contestar a preguntas del tipo: ¿Qué pasó?, ¿quién lo hizo?, ¿cuándo pasó? o ¿qué más se vio afectado?.

1.1.4.2. Garantía de la Calidad del Software (GCS)

La Gestión de la Calidad del Software, o *Software Quality Assurance (SQA)*, se preocupa de asegurar que se consiga el nivel requerido de calidad de un producto software (ver apartado 1.1.3). Ello conlleva la definición de estándares de calidad apropiados y, por supuesto, que estos se sigan rigurosamente [SOM00].

Si nos centramos en la aplicación de la calidad al desarrollo de software puede asegurarse que las especificaciones de un proyecto suelen ser incompletas, a menudo inconsistentes, y si a

ello le unimos además la dificultad que supone evitar ambigüedad en la especificación de algunos requisitos de calidad encontramos que realizar esta especificación resulta ser una tarea dificultosa y muchas veces problemática.

Por tanto, tendremos que a igual que con otras actividades de protección, la GCS tiene a su vez una serie de actividades o tareas a desarrollar que son:

a) Establecimiento de un *plan* de aseguramiento de la calidad del software para el proyecto.
   El plan se desarrolla en la fase inicial del proyecto y es revisado y aprobado por todas las partes implicadas. El plan identifica las evaluaciones, auditorias y/o revisiones a realizar, así como los estándares a seguir o a aplicar, qué procedimientos de información y de corrección de errores se implementaran y como se realizará la documentación.

b) Realizar sistemáticamente el *control* de la calidad, asegurando que los procedimientos y estándares determinados son seguidos por el equipo de desarrollo del software.
   Para ello se realizan una serie de revisiones conocidas como Revisiones Técnicas Formales (RTF) aplicadas en diversos momentos del proceso que tienen como objetivos principales:

   - Descubrir errores.
   - Verificar que el software alcanza los requisitos.
   - Verificar que se siguen los estándares acordados.
   - Cuidar la uniformidad del software desarrollado.
   - Velar por la manejabilidad del proyecto.

Estándar de calidad ISO 9000

La normativa ISO 9000 constituye un conjunto de estándares internacionalmente aceptados para la gestión de la calidad aplicables a un rango de ámbitos de muy diversa tipología, abarcando desde aquellas organizaciones que se dedican a la fabricación de cualquier producto hasta las que proporcionan servicios o productos a partir de procesos altamente industrializados.

La familia ISO 9001 constituye la parte de la normativa ISO 9000 aplicable a organizaciones que realizan tareas de diseño, desarrollo y manutención de productos estableciendo un modelo genérico de proceso de calidad que debe ser instanciado por las propias organizaciones [INC94].

La normativa genérica ISO 9000 no es específica para la industria del software, aunque para ésta dispone de unos apartados determinados:

- ISO 9000-3. *Guidelines for Application of ISO 9001 to the Development, Supply and Maintenance of Software.*
- *ISO 9004-2. Quality Management and Quality System Elements —Part 2—.* En este documento se detallan las directrices a adoptar para proporcionar el soporte a los usuarios por parte del software.

Además de estos fragmentos más específicos del ámbito del desarrollo software, es importante también la parte de la ISO 9001 denominada *Quality Systems-Model for Quality*

Assurance in Design, Development, Production, Installation and Servicing, que describe el sistema de calidad utilizado para mantener el desarrollo de un producto que implique su diseño, tal y como pasa en la implementación de soluciones software.

El aseguramiento de la calidad y los estándares

El enorme esfuerzo que supone adoptar estándares en la gestión de proyectos supone un inmenso trabajo que no se realizaría si realmente no fuese visto como una herramienta verdaderamente potente que apoya la gestión eficaz de los proyectos software obteniendo mejoras apreciables en la calidad de los mismos. Adoptar, por tanto, estándares constituye un factor clave para la gestión efectiva de la calidad.

Uno de los aspectos importantes de definir y aplicar estándares a un proyecto o al desarrollo de un producto —que pueden ser de tipos muy diversos (internacionales, nacionales, organizacionales o simplemente particulares relativos al proyecto) radica en el hecho que define una serie de características comunes que deben exhibir todos los componentes, a la vez que define cómo el proceso software debe ser representado.

Con la adopción de estándares se proporciona un nivel de "encapsulación" que, entre otros factores, evita la repetición de errores que en su día ya fueron detectados y solucionados. Garantizando además la continuidad del proceso aunque cuando éste sea gestionado por nuevos y diferentes equipos humanos.

De todas formas, los equipos de desarrollo software encuentran una serie de problemas cuando aplican los estándares, siendo los siguientes algunos de estos problemas:

- La tecnología avanza a un ritmo tan vertiginoso que parte de estos estándares están rápidamente desfasados.
- El seguimiento de los estándares a menudo se abandona porque implica demasiado esfuerzo burocrático.
- Aunque existen algunas herramientas software destinadas a la gestión de la calidad, en realidad su implantación suele realizarse mediante procesos "manuales", lo que favorece, como hemos visto antes, al abandono de los procesos.

1.1.4.3. Gestión del Riesgo (GR)

> El riesgo es inherente a toda actividad humana y como no podemos eliminarlo tenemos la obligación de reducirlo al máximo.

Se define el riesgo como la posibilidad de sufrir pérdidas o lesiones y la siguiente expresión matemática sirve para definir la exposición del riesgo:

$$ER = P(RI) * P(LO)$$

Donde, ER es la Exposición al Riesgo, P(RI) es la Probabilidad de un Resultado Insatisfactorio y P(LO) representa las pérdidas de las partes afectadas si se produce el resultado insatisfactorio[BOH91].

Ejemplos de resultados insatisfactorios incluyen acabar fuera del plazo previsto, sobrepasar el presupuesto, funcionalidad incorrecta, requisitos no funcionales de difícil comprobación, interfaz de usuario insatisfactorio y de calidad pobre [BOH91].

El desarrollo de software no está ni mucho menos exento de un elevado número de riesgos que pueden hacer fracasar el proyecto. La IS propone la actividad de protección del software conocida como la *Gestión del Riesgo* (GR) que, como su nombre indica, trata de gestionar de manera eficiente todos los aspectos relativos a la identificación de los posibles riesgos con la finalidad de minimizar el impacto final de los mismos.

La Gestión del Riesgo está compuesta por dos tareas principales:

- *La valoración del riesgo*
  Para ello se procede a la identificación de los riesgos posibles para poderlos analizar y posteriormente gestionarlos intentando minimizar su impacto.

  Para identificar los posibles riesgos que un proyecto puede sufrir se procede a la realización de una lista en la que se identifican los posibles factores que pueden comprometer el éxito del proyecto.

  Posteriormente, se analiza para cada ítem de la lista el riesgo concreto valorando la probabilidad de la pérdida y su magnitud.

- *El control del riesgo*
  Dicho control se realiza en base a una planificación que expone las actividades necesarias para poner los factores de riesgo identificados bajo control. Para ello se cuenta con actividades como prototipos, simulaciones, modelados, ajustes, etc. Todos los planes de la gestión deben integrarse para, en lo posible, reutilizar partes de cada uno y descomponer en factores en la planificación total [BOH91].

Con ello los factores de riesgo son eliminados o resueltos en beneficio del proyecto.

Esta faceta de los proyectos software es tan crucial que puede hacer fracasar cualquier proyecto técnicamente perfecto.

Un informe del Parlamento Inglés acerca de la mejora de los proyectos relacionados con la Tecnología de la Información (TI) del año 2000 [PAC99] apunta en una de sus conclusiones que:

La gestión de los proyectos de TI en manos de expertos gerentes es esencial para asegurar que sean entregados a tiempo y de acorde al presupuesto preestablecido. Pero la correcta implementación de los sistemas TI pasa por la imaginación y una perfecta gestión del riesgo junto a una metodología precisa de gestión.

Principios de la Gestión del Riesgo.

El *Carnegie Mellon Software Engineering Institute* (SEI) propone siete principios como marco de trabajo para conseguir una gestión del riesgo efectiva [SEI02]:

Perspectiva global	- Visión del desarrollo del software en el contexto de la definición, el diseño y el desarrollo de grandes niveles de sistemas. - Reconocimiento del valor potencial de la oportunidad y del impacto potencial de los efectos adversos.
Previsión	- Pensando en el mañana, identificando dudas, anticipando resultados potenciales. - Gestionando recursos de proyectos y actividades mientras anticipamos dudas.

Comunicación abierta	- Favorecer el libre flujo de la información entre todos los niveles del proyecto. - Activar todos los niveles comunicación, formales, informales e improvisados. - Uso de procesos que valoren las opiniones personales (llevando el conocimiento y la perspicacia personal para identificar y gestionar el riesgo).
Gestión integrada	- Hacer de la gestión del riesgo una parte vital integrada de la gestión del proyecto. - Adaptando los métodos y técnicas de la gestión del riesgo a la infraestructura y cultura del proyecto.
Proceso continuo	- Vigilancia constante. - Identificación y gestión rutinaria de riesgos durante todas las fases del ciclo de vida del proyecto.
Visión compartida del producto	- Visión mutua del producto basada el propósito común, la propiedad compartida y la comunicación colectiva. - Enfoque sobre los resultados.
Equipo de trabajo (o desarrollo)	- Trabajando cooperativamente hacia la consecución de un objetivo común. - Reunión de talentos, de habilidades y de conocimiento.

Tabla c2_1: Principios de la Gestión del Riesgo detallados por el Instituto de Ingeniería de Software de la Universidad Carnegie Mellon