INTRODUCCIÓN A LA INGENIERÍA DE SOFTWARE

Guillermo Vega Gorgojo – guiveg@tel.uva.es
María Ángeles Pérez Juárez – mperez@tel.uva.es

INDICE

1 ¿ QUÉ ES EL SOFTWARE ?

El software de computadora se ha convertido en el alma mater. Es la máquina que conduce a la toma de decisiones comerciales. Sirve de base para la 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, entretenimientos, productos de oficina, etc., la lista es casi interminable. El software es casi ineludible en un mundo moderno. A medida que nos adentremos en el siglo XXI, será el que nos conduzca a nuevos avances en todo, desde la educación elemental a la ingeniería genética. (Pressman)

El párrafo anterior, casi profético, que aparecía ya en la primera página de la primera edición del libro de Roger S. Pressman: Ingeniería de Software. Un Enfoque Práctico de 1999, sirve para poner de manifiesto la gran importancia del software en el mundo actual y, más aún, el hecho de que la importancia del software crece a medida que el tiempo transcurre puesto que el papel del software en la sociedad continúa su expansión. El software se construye para que sea utilizado potencialmente por cualquier persona en una sociedad del conocimiento como la nuestra. Su importancia radica en que afecta a numerosos, o más bien prácticamente todos, los aspectos de nuestra vida, incluidos los más cotidianos, ya que cada vez más y más sistemas están controlados por software. Las economías de buena parte de los países son cada vez más dependientes del software a la par que aumentan el gasto en desarrollo de software como porcentaje de su Producto Interior Bruto (PIB). En resumen, probablemente, a fecha de hoy, nadie sea capaz de imaginar una sociedad como la nuestra sin software.

Software es el sustantivo genérico que se utiliza para referirse a los programas e información o datos asociados, así como a los medios necesarios para soportar su instalación, operación, reparación y mejora o según la IEEE (Institute of Electrical and Electronic Engineers), el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. Por tanto, el software no son solo programas, sino todos los datos asociados y la configuración de los mismos necesaria para hacer que estos programas operen de manera correcta. Así, un sistema de software consiste en diversos programas independientes, archivos de configuración que se utilizan para ejecutar estos programas, un sistema de documentación que describe la estructura del sistema, la documentación para el usuario que explica cómo utilizar el sistema y sitios web que permitan a los usuarios descargar la información de productos recientes. (Sommerville)

El software de computadora es el producto que los ingenieros de software construyen y después mantienen en el largo plazo. El software se forma con (1) las instrucciones (programas de computadora) que al ejecutarse proporcionan las características, funciones y el grado de desempeño deseados; (2) las estructuras de datos que permiten que los programas manipulen información de manera adecuada; y (3) los documentos que describen la operación y uso de los programas. (Pressman)

2 CARACTERÍSTICAS DEL SOFTWARE VS HARDWARE

Para poder comprender lo que es el software, y consecuentemente la Ingeniería de Software, es importante examinar sus características diferenciadoras con respecto del hardware. Cuando se construye hardware, el proceso de creación (análisis, diseño, construcción, prueba, etc.) se traduce finalmente en una forma física. Así, el boceto inicial, los diagramas formales de diseño y el prototipo de prueba, evolucionan hacia un producto **físico** (chips, tarjetas de circuitos impresos, etc.). Sin embargo, el software es un elemento del sistema que es **lógico**, en lugar de físico. Por tanto, el software tiene unas características considerablemente distintas de las del hardware:

El software se desarrolla, no se fabrica en un sentido clásico. (Pressman)

Aunque existen similitudes entre el desarrollo del software y la fabricación del hardware, ambas actividades son intrínsecamente diferentes. En ambas actividades la buena calidad se adquiere mediante un buen diseño, aunque la fase de construcción del hardware puede introducir problemas de calidad que en el caso del software serían más fácilmente corregibles. Ambas actividades dependen de las personas, pero la relación entre las personas dedicadas y el trabajo realizado es diferente en ambos casos, ya que el desarrollo de software es un proceso más intensivo en capital humano que la fabricación de hardware. En definitiva, ambas actividades están orientadas a la construcción de un producto, pero los enfoques son diferentes.

Un aspecto importante a tener en cuenta al contraponer el hardware frente al software es que el hardware se fabrica, pero el software no. Esto supone que algunos chips fallan desde el principio y es casi seguro que casi todos fallarán pasado un tiempo. Por el contrario, como el software no se fabrica, sino que se desarrolla, si el software se desarrolla correctamente, su duración en el tiempo estaría, a priori, garantizada, es decir, que se podría afirmar algo así como que "si el software sale bien, está bien siempre".

El software no se estropea, pero se deteriora. (Pressman)

La Figura 1 describe la proporción de fallos como una función del tiempo para el hardware. Esa relación, denominada frecuentemente *curva de bañera*, indica que el hardware exhibe relativamente muchos fallos al principio de su vida atribuibles normalmente a defectos del diseño o de la fabricación que no han aflorado hasta el momento. Una vez corregidos estos defectos, la tasa de fallos cae hasta un nivel estacionario generalmente bastante bajo, donde permanece durante un cierto periodo de tiempo. Sin embargo, conforme pasa el tiempo, el hardware empieza a desgastarse y la tasa de fallos se incrementa. El software, al no tener un carácter físico, no es susceptible a las condiciones del entorno que hacen que el hardware se estropee. Por tanto, en teoría, la curva de fallos para el software tendría la forma que se muestra en la Figura 2. Los defectos no detectados antes de entregar el software al usuario, correspondientes típicamente a un error en el diseño o en el proceso mediante el que se tradujo el diseño a código máquina ejecutable, harán que el programa falle durante su primera etapa de vida. Sin embargo, una vez que dichos defectos se corrigen, y suponiendo que al hacerlo no se introducen nuevos errores, la curva se aplanaría tal y como se muestra.

No obstante, la curva idealizada es una simplificación de los modelos reales de fallos del software, ya que, debe tenerse en cuenta que aunque el software no se estropea en un sentido físico, sí se deteriora. Esto que parece una contradicción, puede comprenderse mejor considerando la curva real mostrada en la Figura 2. Durante su vida, el software sufre cambios debido a las labores de mantenimiento. Conforme se realizan los cambios, es probable que se introduzcan nuevos defectos, haciendo que la curva de fallos tenga picos tal y como se observa en la Figura 2. Antes de que la curva pueda volver al estado estacionario original, se solicita otro cambio, haciendo que de nuevo se genere otro pico. Lentamente, el nivel mínimo de fallos comienza a crecer, ya que el software se va deteriorando debido a los cambios. Los métodos de Ingeniería de Software se esfuerzan en reducir la magnitud de los picos así como la inclinación de la curva.

Figura 1: Curva de Fallos del Hardware. (Pressman)

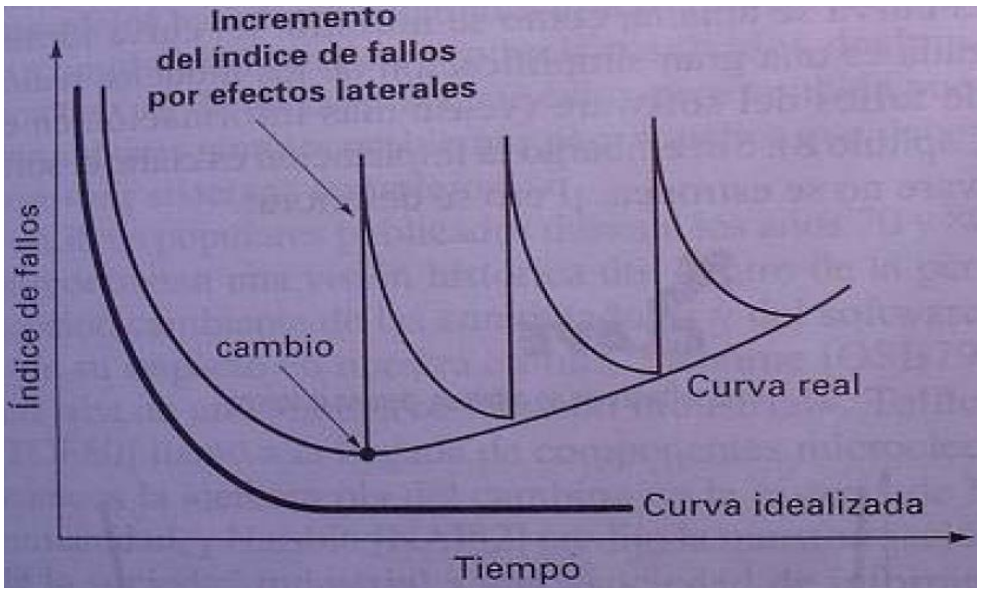


Figura 2: Curvas de Fallos real e idealizada del Software. (Pressman)

Otro aspecto de ese deterioro ilustra otra diferencia existente entre el hardware y el software. Así, cuando un componente de hardware se estropea se puede sustituir por una pieza de repuesto. Sin embargo, no hay piezas de repuesto para el software. Por tanto, el mantenimiento del software tiene una complejidad considerablemente superior al mantenimiento del hardware.

Aunque la industria del hardware tiende a ensamblar componentes, **buena parte del software se construye a medida**. (Pressman)

Considérese la forma en la que se diseña y se construye un determinado hardware de control. El ingeniero de diseño construye un esquema de la circuitería digital, realiza un análisis para asegurar que se consigue la función adecuada y acude entonces a los catálogos de componentes digitales para seleccionarlos y realizar el correspondiente pedido. Existen numerosos componentes estándar (circuitos

integrados, etc.) que utilizan ingenieros mecánicos, eléctricos, etc., cuando diseñan nuevos sistemas. La existencia de estos componentes reutilizables facilita el que los ingenieros puedan concentrarse en los elementos verdaderamente innovadores o diferenciadores de sus diseños. En el mundo del hardware, la reutilización de componentes es una parte natural del proceso de ingeniería.

Sin embargo, la reutilización no se realiza al mismo nivel en el ámbito del software. Un determinado módulo de software debería diseñarse e implementarse para que pudiera volver a utilizarse en diferentes sistemas. Un ejemplo de reutilización de componentes software son las interfaces gráficas de usuario que se construyen con frecuencia a partir de componentes reutilizables que permiten la creación de los elementos necesarios como menús desplegables, etc. A pesar de ello, en la práctica y por diferentes motivos, buena parte del software sigue construyéndose a medida.

3 EVOLUCIÓN DEL SOFTWARE

En los orígenes de las computadoras, el principal esfuerzo se invertía en la mejora del hardware, considerándose el software como algo accesorio y estando éste muy limitado por el hardware. Además, el desarrollo de software era artesanal.

Posteriormente, en los años 70, el coste del hardware se abarata enormemente. Se dispone de hardware de alto rendimiento y gran capacidad de almacenamiento y memoria, nuevos sistemas de entrada y salida, sistemas multiusuario, de tiempo real, etc., lo cual permite disponer de sistemas más sofisticados y complejos. Esto da lugar al problema de que se necesita desarrollar software más complejo, pero se cuenta con las mismas técnicas de desarrollo empleadas hasta el momento lo cual tiene como consecuencia el hecho de que el coste de mantenimiento del software se dispare, así aproximadamente del 50% al 70% del trabajo se hace después de entregar el producto: corregir errores, incorporar mejoras, etc.

Un crecimiento espectacular de los costes del software, un incumplimiento, muchas veces sistemático, de los plazos de entrega y un mar de dudas sobre la calidad de software construido, debido todo ello al hecho de tener que desarrollar software más complejo con técnicas obsoletas, hacen que se produzca una crisis profunda en la industria del software. La necesidad de desarrollar software más complejo hace que surja pareja la necesidad de contar con nuevas técnicas para su desarrollo, se hace necesario el paso de la *artesanía* a la *ingeniería*.

4 LA INGENIERÍA DE SOFTWARE

La **ingeniería** podría definirse como el análisis, diseño, construcción, verificación y gestión de entidades técnicas. Con independencia de la entidad a la que se vaya a aplicar ingeniería, se deben plantear y responder una serie de preguntas:

- ¿Cuál es el problema a resolver?
- ¿Cuáles son las características de la entidad que se utilizará para resolver el problema?
- ¿Cómo se construirá la entidad?
- ¿Cómo se apoyará la entidad cuando los usuarios soliciten correcciones, adaptaciones y mejoras de la misma?
- […]

El profesional en este ámbito recibe el nombre de ingeniero y su actividad se centra en la concreción de una idea en la realidad, puesto que, a través de técnicas, diseños y modelos, y con el conocimiento proveniente de las ciencias, y en ocasiones una buena dosis de inventiva e ingenio, la ingeniería pretende resolver problemas y satisfacer necesidades humanas.

En el caso de la **Ingeniería de Software**, la entidad a construir será el software. La Ingeniería de Software sería la rama de la ingeniería que ofrece métodos y técnicas orientadas a desarrollar y mantener software de calidad. Algunas definiciones formales de Ingeniería de Software son las siguientes:

- Estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas de software. (Zelkovitz)

- Aplicación práctica del conocimiento científico en el diseño y construcción de programas y la documentación asociada requerida para desarrollar y operar[1] y mantenerlos. Se conoce también como desarrollo de software o producción de software. (Boehm)

- Establecimiento y uso de principios sólidos de la ingeniería para obtener de forma económicamente rentable un software confiable y que funcione de modo eficiente en máquinas reales. (Bauer)

- Disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de éste durante su utilización. (Sommerville)

- Disciplina que integra el proceso, las técnicas, y las herramientas para el desarrollo de software. (Pressman)

- Aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación[2] y mantenimiento del software, es decir, la aplicación de la ingeniería al software. (IEEE)[3]

La Ingeniería de Software es por tanto la disciplina que se ocupa del software, enfrentándose al mismo como un producto de ingeniería que requiere planificación, análisis, diseño, implementación, pruebas y mantenimiento. La Ingeniería de Software también incluye las teorías, métodos y herramientas que los profesionales del desarrollo del software deben utilizar. La Ingeniería de Software no solo comprende los procesos técnicos del desarrollo, sino también, los principios más relevantes de dirección y control de dicho proceso, así como el desarrollo de nuevas teorías, métodos y herramientas de apoyo a la producción del software. La Ingeniería de Software persigue mejorar la calidad del software, acortar los tiempos de desarrollo y aumentar la productividad, para lo cual se hace necesario incrementar las posibilidades de reutilización del software. Para lograr este objetivo, los ingenieros de software deben adoptar un enfoque sistemático y organizado en su trabajo y utilizar las herramientas y técnicas más apropiadas dependiendo del problema a resolver, las restricciones del desarrollo y los recursos disponibles.

En ocasiones el término Ingeniería de Software se intercambia con el de **Ciencias de la Computación**, sin embargo, esto no es correcto, puesto que las Ciencias de la Computación tienen que ver con teorías y fundamentos, mientras que la Ingeniería de Software estaría más vinculada a los aspectos prácticos del proceso de desarrollo del software.

---

[1] En el sentido de funcionar.

[2] En el sentido de funcionamiento.

[3] Nótese que aunque la definición de Ingeniería de Software propuesta por la IEEE fue enunciada en 1993, la mayoría de definiciones de dicha disciplina presentadas en esta sección son cronológicamente anteriores. En concreto, se acuñaron en la década de los 70, que fue precisamente cuando se produjo esa profunda crisis en la industria del software que obligó a pasar de artesanía a la ingeniería en el ámbito del software.

Otro término con el que a menudo se intercambia el de Ingeniería de Software es el de **Ingeniería de Sistemas**, sin embargo, nuevamente es importante matizar esto, puesto que Ingeniería de Sistemas e Ingeniería de Software son dos términos íntimamente relacionados, pero no intercambiables [ver Figura 3].

Un **sistema** es una colección de componentes interrelacionados que trabajan conjuntamente para cumplir algún objetivo. La Ingeniería de Sistemas consiste en la actividad de especificar, diseñar, implementar, validar, distribuir y mantener sistemas como un todo. Por este motivo, los ingenieros de sistemas no solo están relacionados con el software, sino también con el hardware y con las interacciones del sistema con los usuarios y su entorno. Es decir, que la Ingeniería de Sistemas es un campo de trabajo más amplio que la Ingeniería de Software ya que la Ingeniería de Sistemas tiene que ver con todos los aspectos del desarrollo de sistemas basados en computadoras, incluyendo hardware y software, mientras que la Ingeniería de Software se centraría únicamente en una parte de todo el proceso.

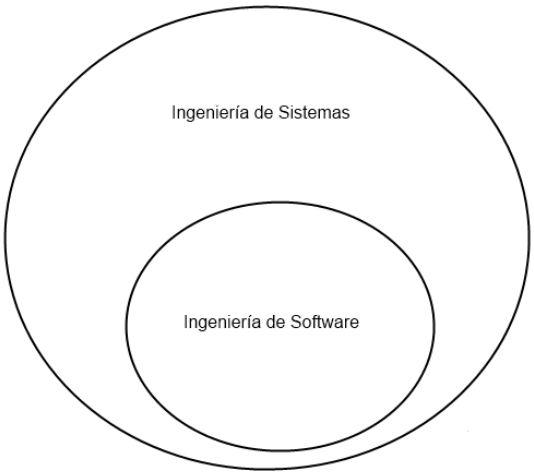


Figura 3: Ingeniería de Sistemas vs Ingeniería de Software.

El trabajo que típicamente se asocia a la Ingeniería de Software se puede dividir en tres fases genéricas: definición, desarrollo y mantenimiento, con independencia del área de aplicación, tamaño o complejidad del proyecto.

Definición. La fase de definición se centra en el qué. Así, durante esta fase se intenta identificar qué información ha de ser procesada, qué comportamiento se espera del sistema, qué restricciones existen, etc. En definitiva, las tareas principales serán la captura de información, la planificación del proyecto software y el análisis de los requisitos.

Desarrollo. La fase de desarrollo se centra en el cómo. Así, durante esta fase se intenta definir cómo se diseñarán las estructuras de datos, cómo se traducirá el diseño a código, etc. En definitiva, las tareas principales serán el diseño del software, la generación de código y las pruebas del software.

Mantenimiento. La fase de mantenimiento se centra en gestionar los cambios de diferentes tipos que podrá sufrir el software:

- **Corrección**. A pesar de llevar a cabo actividades para garantizar la calidad del software, es muy probable que se descubran errores. El mantenimiento correctivo tendrá por objetivo cambiar el software para corregir sus defectos.

- **Adaptación**. Con el paso del tiempo, es probable que cambie el entorno original para el que se desarrolló el software. El mantenimiento adaptativo modifica el software para acomodarlo a los cambios de su entorno externo.[4]

- **Mejora**. Conforme se utilice el software, el usuario puede descubrir funciones adicionales que producirían beneficios. El mantenimiento perfectivo lleva al software más allá de sus requisitos funcionales originales.

- **Prevención**. El software se deteriora debido al cambio, por este motivo, el mantenimiento preventivo también denominado **reingeniería del software**, intenta realizar cambios en el software para que este se pueda corregir, adaptar y mejorar más fácilmente.

Además de estas actividades de mantenimiento, los usuarios de software requieren de una asistencia continuada que se puede proporcionar mediante diversos mecanismos: ayuda en línea, etc. Ya que, cuando se emplea el término mantenimiento se entiende que es una actividad que va mucho más que una simple corrección de errores.

Las actividades descritas en las diferentes fases se complementan con un número de actividades de carácter protector como el seguimiento y control del proyecto de software, la garantía de calidad del software, la documentación, la gestión de reutilización o la gestión de los riesgos potenciales.

5 EL PROCESO DEL SOFTWARE

Al hablar de software puede hacerse hincapié tanto en el producto final (**el software como un producto**) como en su proceso de desarrollo (**el proceso del software**). Esta sección se centra en la vertiente del software como proceso.

El proceso del software y la Ingeniería de Software son términos íntimamente relacionados, así, un proceso de software define el enfoque que se aplica cuando el software es tratado utilizando una aproximación ingenieril tal y como hace la Ingeniería de Software. Pero la Ingeniería de Software necesita también de las técnicas, incluyendo la notación que estas emplean, y de las herramientas vinculadas al proceso.

Podría decirse que la Ingeniería de Software necesita de un **método**, puesto que se hace necesario establecer un enfoque sistemático y disciplinado para llevar a cabo un desarrollo software. Definiciones posibles para el término método en este contexto serían las siguientes:

- Una metodología de Ingeniería de Software es un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas. (James Rumbaugh et al.)

- Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software. (Mario Piattini et al.)

[4] Por ejemplo, cambios en el sistema operativo, cambios en las reglas del negocio, etc.

Figura 4: ¿Qué necesita la Ingeniería de Software?

En resumen, las componentes del método necesario en la Ingeniería de Software serían un proceso, una serie de técnicas con su notación y un conjunto de herramientas. El proceso define el marco de trabajo y permite un desarrollo racional de la Ingeniería de Software. Las técnicas indican cómo construir técnicamente el software e incluyen técnicas de modelado. Y las herramientas proporcionan el soporte automático o semiautomático para el proceso y para las técnicas. En definitiva, la Ingeniería de Software requiere de un Proceso[5], de una serie de técnicas que hacen uso de una Notación[6] y de una serie de Herramientas[7]. [ver Figura 4]

Proceso:

Cuando se trabaja para construir un producto o un sistema software, es importante seguir una serie de pasos predecibles, algo así como una hoja de ruta que ayude a obtener un resultado con el nivel de calidad deseado. La hoja de ruta a seguir es lo que se denomina proceso del software.

Los ingenieros de software y sus gestores adaptarán el proceso del software a las necesidades de cada proyecto del que deban hacerse cargo. Además, los clientes que hayan solicitado el software deberán, idealmente, colaborar en dicho proceso. Desde el punto de vista del ingeniero de software, se podría decir que los productos obtenidos al finalizar el proceso serán programas, documentos y datos que se producirán como consecuencia de las actividades de Ingeniería de Software definidas por el proceso.

La Ingeniería de Software necesita de un proceso porque este proporciona una aproximación metódica, documentada y reproducible. El proceso del software es importante porque proporciona estabilidad, control y organización a una actividad que, de otra forma, podría volverse caótica. El proceso facilita optimizar el desarrollo, acudiendo a la paralelización de tareas y a la reutilización de resultados cuando resulte posible, así como minimizar los riesgos que puedan presentarse. Además, existen diversos mecanismos de evaluación del proceso del software que permiten a las organizaciones determinar la madurez de su proceso del software.

[5] En esta asignatura el proceso que se seguirá será el Proceso Unificado (UP, Unified Process) de Desarrollo de Software.

[6] En esta asignatura la notación a utilizar será el Lenguaje de Modelado Unificado (UML, Unified Modelling Language).

[7] En esta asignatura la herramienta CASE UML a utilizar será Astah.

Sin embargo, la calidad a largo plazo del producto que se está construyendo, y que será necesario mantener, es el mejor indicador de la eficiencia del proceso que se está utilizando.

El proceso del software se puede caracterizar estableciendo un marco de trabajo común que defina actividades aplicables a todos los proyectos de software, con independencia de su tamaño, complejidad o dominio de aplicación. Así como una serie de tareas propias de la Ingeniería de Software vinculadas a cada actividad definida que permiten que el proceso se adapte a las características concretas del proyecto de software que se desarrolle. Y, finalmente, unas actividades de protección, tales como la garantía de calidad del software, que sean independientes de cualquier actividad del marco de trabajo y que estarán presentes durante todo el proceso.

Las actividades definidas en el marco del proceso del software se denominan, en algunos textos, Áreas Clave de Proceso (ACPs), describen las funciones propias de la Ingeniería de Software (planificación del proyecto de software, gestión de requisitos, etc.), forman la base del control de la gestión de proyectos de software y establecen el contexto en el que se aplica la metodología, se obtienen productos del trabajo (modelos, documentos, etc.), se establecen hitos, etc. Cada ACP se describe identificando sus objetivos, los requisitos impuestos junto a los objetivos, las capacidades o elementos de carácter organizativo y técnico necesarios y las tareas específicas que deben llevarse a cabo para cumplir los objetivos, los métodos para implementar dichas tareas y los métodos para verificar su implementación.

Técnicas y Notación:

Las **técnicas** indican cómo construir técnicamente el software. Las técnicas de la Ingeniería de Software incluyen actividades de modelado y otras técnicas descriptivas en las que se hace uso de una **notación**. El lenguaje natural que el ser humano maneja habitualmente es vago, ambiguo e interpretable según intereses, y esto hace necesaria una notación estándar que resulte visual y adecuada para dialogar con el cliente, a la par que detallada y sin ambigüedades para intercambiar información con los ingenieros de software.

Herramientas:

Las **herramientas** proporcionan un enfoque automático o semiautomático para el proceso y para las técnicas, estableciéndose un sistema de soporte para el desarrollo del software denominado Ingeniería de Software Asistida por Ordenador (CASE, Computer Aided Software Engineering). Las herramientas son necesarias porque facilitan el seguir el proceso y el utilizar las técnicas y su notación, ayudan a la gestión de versiones o reducen el trabajo de codificación[8]. [ver Figura 5]

[8] El término codificación en este contexto, se refiere a la generación o creación del código.

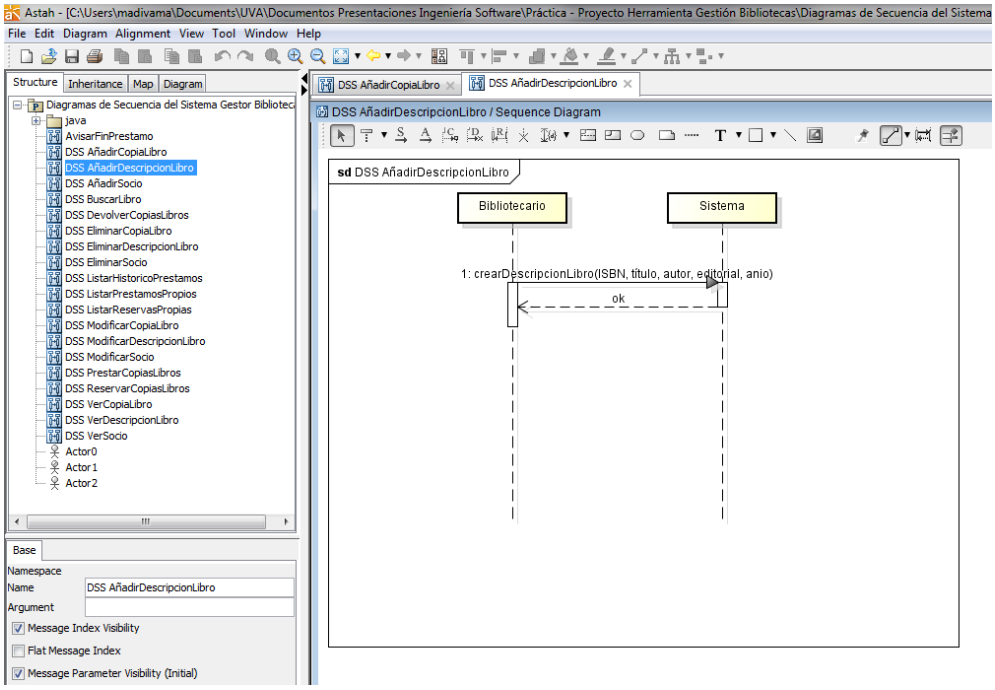


Figura 5: Vista de la herramienta CASE Astah.

6 MODELOS DE PROCESO DEL SOFTWARE

6.1 MODELO DE PROCESO O CICLO DE VIDA DEL SOFTWARE

Para acometer los proyectos de software, los ingenieros de software deben incorporar una estrategia de desarrollo que acompañe a las técnicas y herramientas y a las fases genéricas del proceso del software. Esta estrategia a menudo se denomina **modelo de proceso** o **ciclo de vida** del software o **paradigma** de Ingeniería de Software. Se seleccionará un modelo de proceso u otro según la naturaleza del proyecto y de la aplicación, las técnicas y las herramientas a utilizar, y los controles y entregas que se requieran. Un modelo de proceso proporciona una vista de las actividades que se llevan a cabo durante su desarrollo, e intenta determinar el orden de las etapas involucradas y proporcionar unos criterios para avanzar de unas a otras. Por tanto, definir un ciclo de vida permite llevar un mayor control sobre las tareas, evitando que estas se vayan eligiendo y realizando de manera desordenada, según parezca que van surgiendo necesidades.

6.2 EL MODELO CLÁSICO O MODELO EN CASCADA

El **Modelo Clásico**, en ocasiones denominado Ciclo de Vida Básico o Modelo en Cascada[9] [ver Figura 6], es un modelo lineal que sugiere un enfoque sistemático y secuencial para el desarrollo de

[9] En los textos ingleses al Modelo Clásico o Modelo en Cascada se le denomina `Waterfall Model`.

software que progresa con el Análisis, Diseño, Codificación, Pruebas y Mantenimiento. En concreto, el modelo lineal secuencial comprende las siguientes actividades:

Captura de Requisitos (Análisis). Como el software por lo general siempre forma parte de un sistema más grande, el trabajo comienza estableciendo los requisitos del sistema y asignando al software algún subconjunto de estos requisitos. Para comprender la naturaleza del software que debe construirse, el ingeniero de software debe comprender el dominio al que pertenece dicha aplicación software, así como la función requerida, comportamiento, rendimiento e interconexión. En definitiva, la etapa de Análisis implica entender:

- El dominio de aplicación del software.
- Qué debe hacer el software, es decir, su funcionalidad.
- Otras restricciones: eficiencia, uso de recursos, seguridad, conectividad, etc.

Diseño del Sistema. El diseño del software es un proceso que se centra en las estructuras de datos, la arquitectura del software y el detalle procedimental (algoritmo). El proceso del diseño traduce requisitos en una representación del software donde se pueda evaluar su calidad antes de que comience la codificación. En definitiva, la etapa de Diseño implica pensar en cómo se va a hacer el software, lo cual supone organizar la arquitectura del software y detallar cómo los módulos implementarán la funcionalidad.

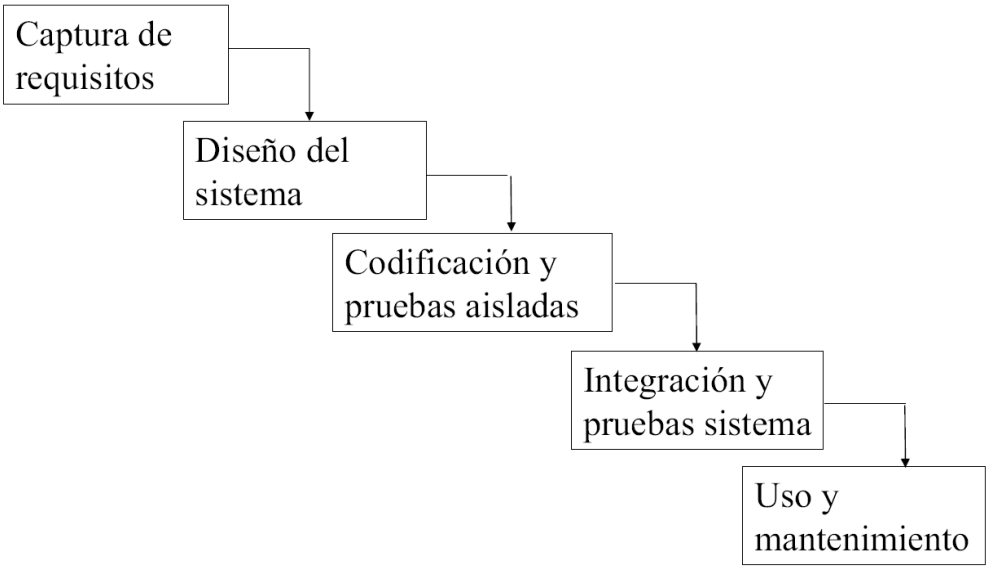


Figura 6: Modelo Clásico o Modelo en Cascada – lineal secuencial –.[10]

Codificación y pruebas aisladas. El diseño se debe traducir en una forma legible por la máquina. En la fase de Codificación se lleva a cabo la generación del código. Si el diseño se lleva a cabo de una forma detallada, la generación de código se podrá realizar de una forma bastante mecánica. En esta etapa se codifica cada módulo y se prueba por separado.

Integración y Pruebas de Sistema. Una vez que se ha generado todo el código, comienzan las pruebas del programa. El proceso de pruebas se centra tanto en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, como en los procesos externos

[10] En los textos ingleses las fases del Modelo Clásico o en Cascada se denominan `Analysis`, `Design`, `Implementation`, `Testing` y `Deployment`.

funcionales, lo que supone realizar las pruebas necesarias para asegurar que las entradas producen resultados reales de acuerdo con los planificados. En definitiva, esta etapa consiste en juntar todos los módulos y probarlos, para asegurarse de que el programa cumple con la funcionalidad para la cual se ha diseñado, y que, por tanto, se satisfacen los requisitos de partida.

Uso y Mantenimiento. El software probablemente sufrirá cambios después de ser entregado al cliente. Se realizarán cambios porque se han encontrado errores, porque el software debe adaptarse a los cambios de su entorno externo (cambio en el sistema operativo, etc.) o porque el cliente requiere mejoras (funcionales, de rendimiento, etc.). En esta etapa se vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo. En definitiva, en esta etapa se debe instalar el software en las máquinas del cliente o en el servidor correspondiente y mantenerlo adecuadamente.

El modelo clásico es el paradigma más antiguo y extensamente utilizado en la Ingeniería de Software y resulta un enfoque razonable cuando los requisitos se han entendido correctamente y se trata de un sistema simple. Como principales ventajas cabe señalar el hecho de que se trata de un proceso muy bien definido y que consta de tareas repartibles a individuos o a grupos.

Sin embargo, es un modelo no exento de críticas y su eficacia ha sido puesta en duda. Entre los problemas que se encuentran algunas veces en el modelo clásico se cuentan los siguientes:

- Los proyectos reales raras veces siguen el esquema secuencial que propone el modelo[11]. A menudo es difícil que el cliente exponga explícitamente todos los requisitos. El modelo lineal secuencial lo requiere y tiene dificultades a la hora de acomodar la incertidumbre natural existente al comienzo en muchos proyectos. Así, por ejemplo, es casi imposible conocer, en el instante inicial, un sistema complejo y de grandes dimensiones al detalle, y este modelo requiere que esto sea así antes de poder avanzar desde la fase de Análisis hacia la fase de Diseño.

- El cliente debe tener paciencia. Una versión de trabajo del software no estará disponible hasta que el proyecto esté muy avanzado. Y en grandes proyectos, cada fase puede ser considerablemente larga. Además, un error grave puede ser desastroso si no se detecta hasta que se revise el software. De hecho, este modelo hace que el riesgo se traslade a etapas posteriores del proceso, ya que los problemas más importantes se suelen identificar en etapas avanzadas, especialmente durante la etapa de Integración y pruebas del sistema, cuando, irónicamente el coste de rectificar errores aumenta exponencialmente a medida que el tiempo pasa. [ver Figura 7]

[11] El modelo original en cascada propuesto por Winston Royce preveía bucles de realimentación, no obstante, la gran mayoría de las organizaciones que aplican este modelo de proceso lo hacen como si fuera estrictamente lineal, lo cual implica que cada etapa debe completarse totalmente antes de que la siguiente pueda comenzar.

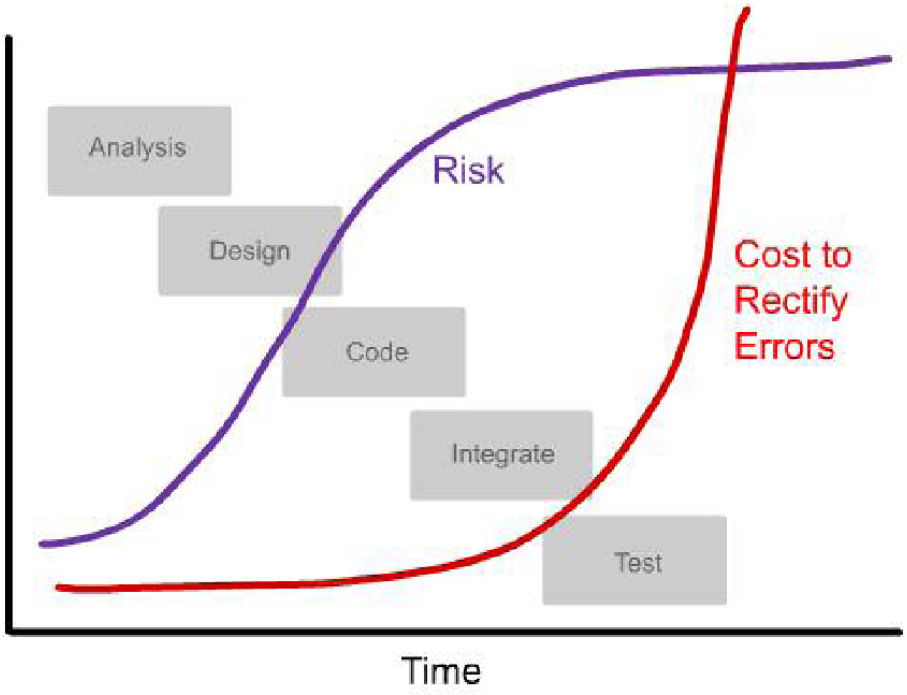


Figura 7: Representación del Riesgo y del Coste de Rectificar Errores en el Modelo Clásico o Modelo en Cascada – lineal secuencial –. (Ariadne)

En el Modelo Clásico o Modelo en Cascada, en cada etapa se pueden descubrir fallos de la etapa anterior y, en esta situación ¿qué hacer?, ¿se para todo el proceso y se vuelve al principio? o ¿se sigue con los fallos hasta el final? Además, está el problema de cómo proceder si se descubren nuevos requisitos o si los ya existentes cambian.

El paradigma del ciclo de vida clásico tiene un lugar importante en el ámbito de la Ingeniería de Software, porque, pese a sus puntos débiles es significativamente mejor que un enfoque al azar para el desarrollo del software. No obstante, falla cuando la complejidad y el tamaño del proyecto al que debemos enfrentarnos se incrementa. Es decir, este modelo de proceso se adecua a proyectos suficientemente pequeños. La definición de suficientemente pequeño es obviamente subjetiva, pero se referiría a proyectos que puedan ser abordados por equipos reducidos donde cada miembro del equipo pueda conocer los diferentes aspectos del sistema y donde la duración del ciclo de vida se mantenga en unos límites de tiempo razonables.

6.3 EL MODELO EN ESPIRAL

Un enfoque alternativo al Modelo en Cascada es el Modelo en Espiral[12]. Este modelo propone abordar el proyecto de desarrollo software al que nos enfrentemos en una serie de ciclos de vida cortos (Análisis, Diseño, Codificación y Pruebas)[13], al final de cada uno de los cuales se debe producir una versión ejecutable del software en cuestión. [ver Figura 8]

Este enfoque iterativo permite recibir retroalimentación del cliente de forma más frecuente, en concreto, al final de cada ciclo, lo cual permite identificar y abordar problemas antes de haber llegado

[12] En los textos ingleses al Modelo en Espiral se le denomina `Spiral Model`.

[13] En los textos ingleses las fases del Modelo en Espiral se denominan `Analysis`, `Design`, `Code` y `Test`.

demasiado lejos en el proyecto. El riesgo puede manejarse de forma más adecuada ya que los ciclos más arriesgados, en los que, por ejemplo, haya que trabajar con una tecnología nueva aún no validada, pueden abordarse primero. Este modelo también permite incorporar de forma más sencilla los cambios que pudieran tener lugar en los requisitos. Sin olvidar el hecho de que el producir versiones de software tras cada ciclo, que no debería ser excesivamente largo, imprime moral al equipo de desarrolladores que ven el fruto de su trabajo y satisface igualmente al cliente.

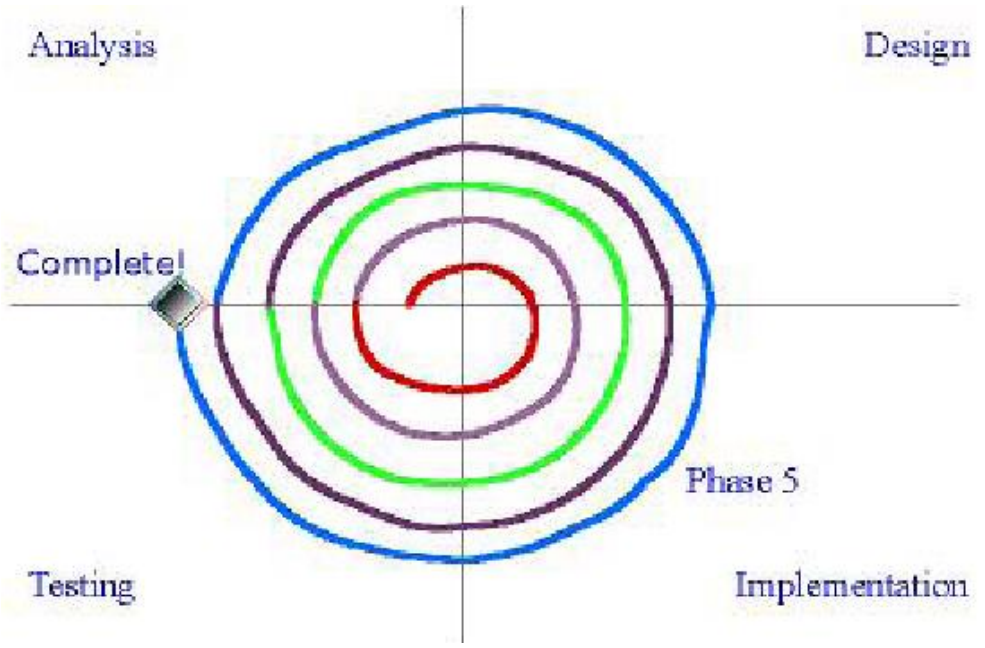


Figura 8: Representación de un proceso espiral de desarrollo software en el que se ha dividido el proyecto en cinco fases, cada una basada en la anterior y de forma que al final de cada fase se disponga de una versión de software ejecutable. (Ariadne)

No obstante, este modelo tampoco está exento de críticas, ya que el proceso en este caso es más difícil de gestionar que en el Modelo en Cascada en el cual pueden utilizarse de manera sencilla técnicas de gestión de proyectos clásicas como los Diagramas de Gantt[14].

6.4 El Marco de Trabajo Iterativo e Incremental

Para contrarrestar las desventajas del Modelo en Espiral se propone un enfoque similar, pero más formal, denominado Marco de Trabajo Iterativo e Incremental[15]. Este enfoque es por tanto una extensión lógica del Modelo en Espiral pero más rigurosa. Este marco de desarrollo divide el trabajo a desarrollar en cuatro grandes fases: Arranque o Inicio, Elaboración, Construcción y Transición[16] [ver Figura 9], que no deben confundirse con las fases del Modelo Clásico o Modelo en Cascada, aunque también se desarrollen de forma secuencial. La Fase de Construcción incorpora habitualmente una serie

[14] Herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un periodo determinado.

[15] En los textos ingleses al Marco de Trabajo Iterativo e Incremental se le denomina `Iterative Incremental Framework`.

[16] En los textos ingleses las fases del Marco de Trabajo Iterativo e Incremental se denominan `Inception`, `Elaboration`, `Construction` y `Transition`.

de iteraciones[17] similares al proceso propuesto en el Modelo Clásico o Modelo en Cascada [ver Figura 10] y es, habitualmente, la de mayor duración [ver Figura 11].

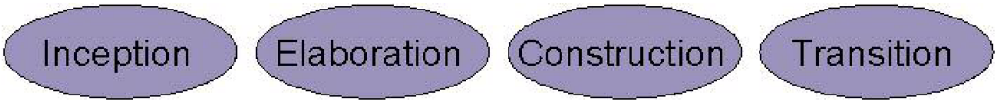


Figura 9: Fases del Marco de Trabajo Iterativo e Incremental. (Ariadne)

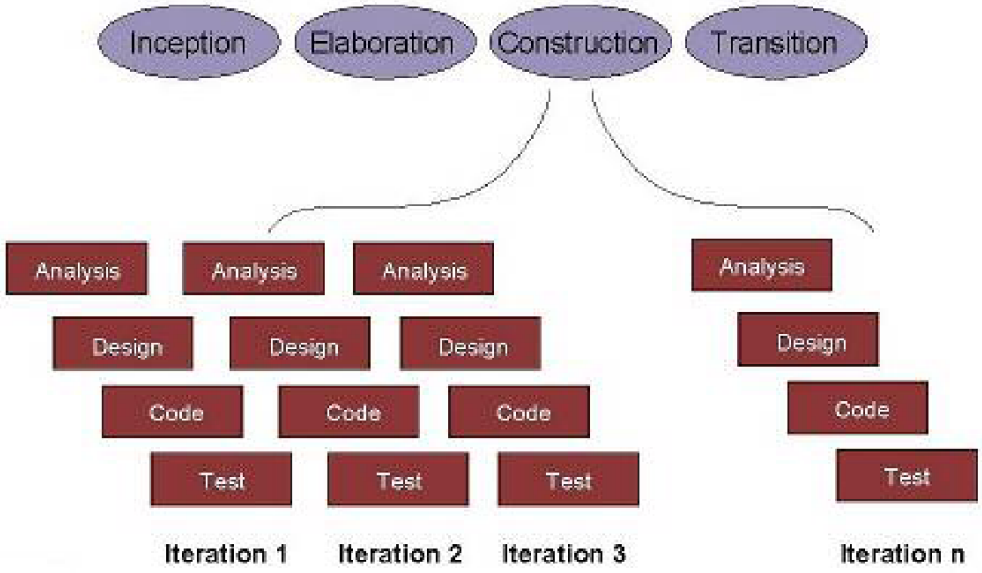


Figura 10: La Fase de Construcción del Marco de Trabajo Iterativo e Incremental comprende una serie de iteraciones similares al proceso propuesto por el Modelo Clásico o Modelo en Cascada. (Ariadne)

El ejemplo más famoso de un Ciclo de Vida Iterativo e Incremental es el Proceso Unificado de Desarrollo de Software (Unified Software Development Process) que será el que se aplique para el trabajo en la presente asignatura por ser el Modelo de Proceso de Software dominante en la Ingeniería de Software moderna.

[17] En algunos textos, a las **iteraciones** del Marco de Trabajo Iterativo e Incremental se les denomina **incrementos**.

El resto de las fases podrían incorporar interacciones, pero en la práctica es poco habitual.

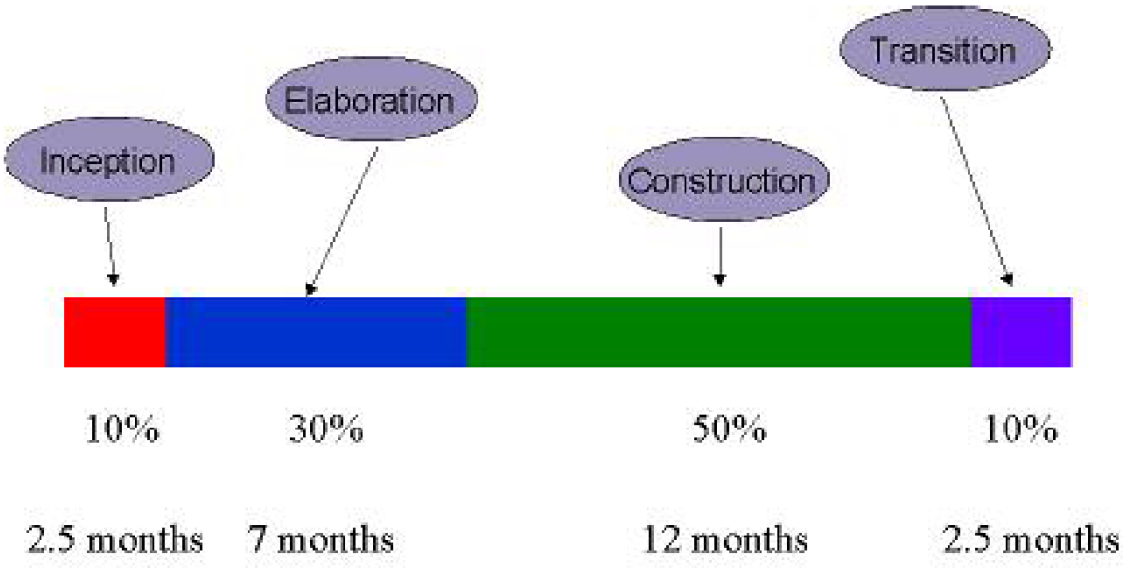


Figura 11: Posible duración de cada fase en el Marco de Trabajo Iterativo e Incremental para un proyecto de dos años de duración. La Fase de Construcción que comprende una serie de iteraciones similares al proceso propuesto por el Modelo Clásico o Modelo en Cascada será típicamente la de mayor duración. (Ariadne)

6.5 El Proceso Unificado de Desarrollo de Software

El Proceso Unificado de Desarrollo de Software, que fue propuesto por Rumbaugh, Booch y Jacobson, pretende ser un marco de proceso genérico y extensible que pueda ser adaptado a organizaciones, equipos o proyectos específicos. Sus principales características son las siguientes:

Utiliza el Lenguaje de Modelado Unificado (UML, Unified Modelling Language) como lenguaje de modelado. [ver sección 7 El Lenguaje de Modelado Unificado]

Es un marco de desarrollo iterativo e incremental. Esto supone que es un marco de trabajo que propone múltiples **ciclos** cada uno comprendiendo las **fases** de Arranque o Inicio, Elaboración, Construcción y Transición. En este marco de desarrollo, cada una de las fases anteriores comprende a su vez una serie de iteraciones que ofrecen como resultado un incremento del producto desarrollado, que añade o mejora las funcionalidades del sistema en desarrollo.

Durante cada una de estas **iteraciones** se realizarán a su vez una serie de **actividades** que, tal y como es sabido, recuerdan a las definidas en el Modelo Clásico o Modelo en Cascada: Requisitos, Análisis, Diseño, Implementación y Pruebas (e Implantación).[18]

Aunque todas las iteraciones suelen incluir trabajo en casi todas estas actividades, el grado de esfuerzo dedicado a cada una de ellas varía a lo largo del proyecto. Así, por ejemplo, la fase de Arranque o Inicio se centrará más en la definición de Requisitos y en el Análisis, actividades que durante la fase de

[18] En definitiva, el Proceso Unificado comprende múltiples ciclos de cuatro fases (Arranque o Inicio, Elaboración, Construcción y Transición), cada fase a su vez con múltiples iteraciones, y en cada iteración se podrán desarrollar cinco tipos de actividades (Requisitos, Análisis, Diseño, Implementación y Pruebas (e Implantación)). Nótese, no obstante, que algunos autores proponen que solo se incluyan iteraciones en la fase de Construcción, de forma que las fases de Arranque o Inicio y Elaboración se mantengan sencillas.

Construcción quedarán relegadas a favor, por ejemplo, de las actividades de Implementación y Pruebas [ver Figura 12].

Si una iteración cumple sus metas, publicando una nueva versión del producto que implemente ciertos **Casos de Uso**[19], el desarrollo continúa con la siguiente. Cuando no las cumple, los desarrolladores deben revisar sus decisiones previas y probar un nuevo enfoque.

El hecho de que sea un proceso iterativo e incremental reduce riesgos, ya que se puede comenzar trabajando sobre los aspectos menos claros para volver sobre ellos hasta encontrar una solución adecuada, y permite acomodar requisitos cambiantes.

Está guiado por los Casos de Uso. Un sistema software se crea para dar respuesta a las necesidades de sus usuarios por lo que, para construir un sistema exitoso, se debe conocer qué es lo que realmente quieren y necesitan. El término usuario no se refiere solamente a los usuarios humanos sino también a otros sistemas software, es decir, se refiere a algo o alguien que interactúa con el sistema a desarrollar.

En el Proceso Unificado de Desarrollo de Software, los Casos de Uso permiten capturar los denominados **Requisitos Funcionales**[20] a la par que definir el objetivo y contenido de las iteraciones. La idea es que cada iteración tome un conjunto de Casos de Uso y desarrolle el camino de manera completa a través de las distintas actividades: Diseño, Implementación, etc. En cada iteración, los desarrolladores identifican y especifican los Casos de Uso relevantes, crean el diseño, lo implementan en componentes y verifican que los componentes satisfacen los Casos de Uso.

El que el Proceso Unificado de Desarrollo de Software sea un proceso guiado por Casos de Uso permite capturar el valor añadido del sistema software para el cliente, es decir, especificar qué proporciona el sistema software al cliente en un lenguaje que el cliente entienda.

También se cuenta con una guía para todo el proceso, así los Casos de Uso inician el proceso y son el nexo entre las actividades desarrolladas en las iteraciones ya que aparecen representados de forma subyacente en todos los artefactos utilizados para el modelado. Además, la utilización de Casos de Uso facilita el desarrollo iterativo, la realización de documentación y constituyen el plan de pruebas.

Está Centrado en la Arquitectura. El concepto de arquitectura del software[21] involucra los aspectos estáticos y dinámicos más significativos del sistema, dando una perspectiva completa y describiendo los elementos más importantes. La arquitectura surge de los propios Casos de Uso, sin

[19] Los Casos de Uso son una técnica del Lenguaje de Modelado Unificado que permite capturar información acerca de cómo trabaja un sistema actualmente y de cómo se desea que trabaje. Además, se trata de una técnica que pone énfasis en la relación del sistema con el exterior. Permite la captura de requisitos (funcionales) y sirve igual para un enfoque procedimental que para una filosofía orientada a objetos. Un Caso de Uso puede definirse como una interacción típica entre el usuario y el sistema que captura una función visible al usuario y logra un objetivo del usuario que puede ser grande o pequeño. Los Casos de Uso se pueden representar visualmente a través de los Diagramas de Casos de Uso, en los que además de los Casos de Uso como tal, representados por un óvalo con su nombre, aparecen los Actores de dichos Casos de Uso representados por un monigote con su nombre. A través del Diagrama de Casos de Uso del sistema se puede representar quiénes usan el sistema: roles de personas, máquinas u otros sistemas software, y saber qué quieren del sistema, además de determinar qué está dentro del sistema y qué queda fuera. Véase más información en el documento dedicado a este tema de forma específica.

[20] Un Requisito Funcional es una característica requerida del sistema que expresa una capacidad de acción del mismo, es decir, una funcionalidad, generalmente expresada en una declaración en forma verbal. Véase más información sobre el concepto de Requisito Funcional en el documento dedicado a este tema de forma específica.

[21] El Proceso Unificado de Desarrollo de Software asume que no existe un diagrama único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples diagramas o vistas que definen la arquitectura de software de un sistema, de igual forma que cuando se construye un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc.

embargo, también está influenciada por muchos otros factores, como la plataforma en la que se ejecutará, el uso de estándares o los **Requisitos No Funcionales**[22]. Los Casos de Uso deberán poder acomodarse en la arquitectura que debe ser lo bastante flexible para realizar todos los Casos de Uso, hoy y en el futuro. De hecho, arquitectura y Casos de Uso deberían evolucionar en paralelo.

El Proceso Unificado de Desarrollo de Software es un proceso centrado en la arquitectura porque los Casos de Uso no resultan suficiente. La arquitectura permite organizar cómo el software cumplirá los requisitos, decidiendo los subsistemas y sus interfaces, qué hace cada subsistema y si un subsistema puede construirse a partir de otros subsistemas.

Está Enfocado a los Riesgos. Para disminuir la posibilidad de fallo en las iteraciones o incluso la posibilidad de cancelación del proyecto, se deben llevar a cabo sucesivos análisis de riesgos durante todo el proceso. Los riesgos críticos deberían identificarse en una etapa temprana del ciclo de vida del sistema software, en este sentido, los resultados de cada iteración de las diferentes fases deberían seleccionarse en un orden que asegure que dichos riesgos se considerarán en primer lugar.

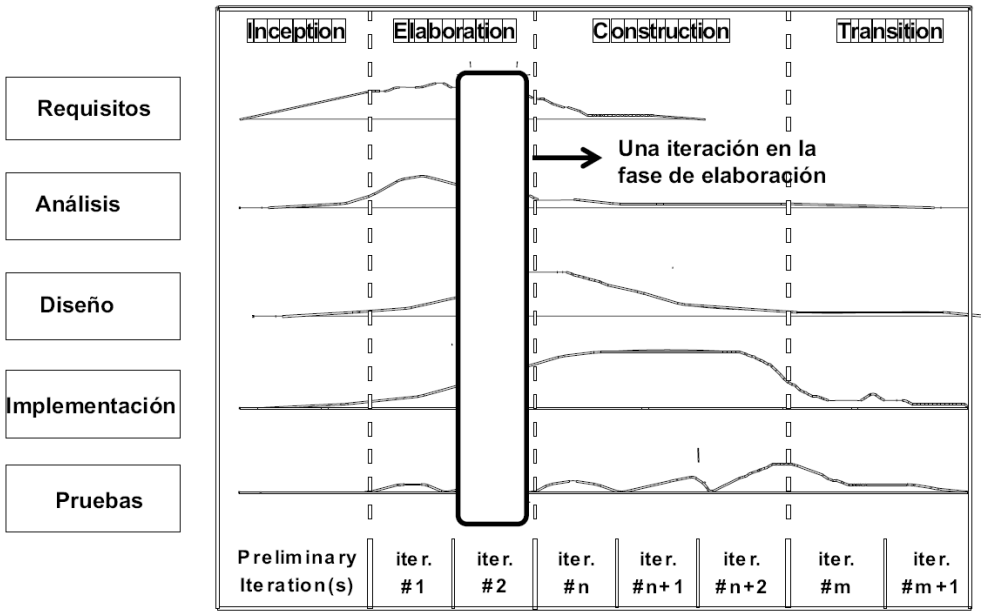


Figura 12: Ciclo de Vida especificado por el Proceso Unificado de Desarrollo de Software: Fases, Iteraciones y Actividades. El Proceso Unificado de Desarrollo de Software es iterativo e incremental. Es un proceso organizado en múltiples ciclos de cuatro fases, cada una de ellas a su vez con múltiples iteraciones y cada iteración con cinco posibles actividades.

Como ya se ha comentado, el Proceso Unificado de Desarrollo de Software se organiza en una serie de ciclos que constituyen la vida de un sistema. Cada ciclo se compone de cuatro fases: Arranque o Inicio, Elaboración, Construcción y Transición, y dentro de cada una de ellas, los desarrolladores pueden descomponer adicionalmente el trabajo en iteraciones, cada una de ellas con cinco posibles actividades: Requisitos, Análisis, Diseño, Implementación y Pruebas (e Implantación), con sus incrementos resultantes. Además, periódicamente se alcanzan hitos en forma de la disponibilidad de un determinado artefacto o diagrama de modelado[23] y/o documento de especificación. [ver Figura 12]

[22] Un Requisito no Funcional es una característica requerida del sistema, del proceso de desarrollo o del servicio prestado, que típicamente señala una restricción de este. Véase más información sobre el concepto de Requisito Funcional en el documento dedicado a este tema de forma específica.

[23] El Lenguaje de Modelado Unificado es un lenguaje gráfico de modelado que proporciona la sintaxis para describir los elementos principales de un sistema software. Estos elementos principales de un sistema software,

6.5.1 Fases del Proceso Unificado de Desarrollo de Software

A continuación se describe brevemente cada una de las Fases del Proceso Unificado de Desarrollo de Software:

Fase de Arranque o Inicio. En esta fase, que terminará al alcanzar el hito de los objetivos del desarrollo, se realizarán típicamente las siguientes tareas:

- Desarrollar una descripción del producto final[24], así como un adecuado **análisis del negocio** al que este pertenece.
- Realizar una identificación inicial de riesgos y **estudiar la viabilidad**, para poder determinar si es factible o no llevar a cabo el proyecto.
- **Capturar los Requisitos iniciales**. Establecer las principales funciones del sistema para los usuarios más importantes.

Algunos de los entregables que podrían producirse en esta fase son los siguientes documentos[25]:

- Documento de **Visión** del sistema software que básicamente incluye la **Descripción Informal** del sistema software, acompañada, de forma muy frecuente, por el diseño del **prototipo de la Interfaz de Usuario**.
- **Glosario** del proyecto software con especial atención a todo lo relativo a la lógica del dominio o negocio abordado.
- **Catálogo de Especificación de Requisitos** inicial del sistema software.
- **Diagrama de Casos de Uso** inicial.
- Documento de **Evaluación de Riesgos** inicial del proyecto software. Incluirá los riesgos técnicos, de recursos o del negocio vinculados al proyecto, y que formule asimismo un plan con medidas preventivas y correctivas para abordar cada uno de los riesgos identificados en caso de que estos se presenten. Es decir, no se trata de una simple identificación de riesgos potenciales sino de prever una adecuada gestión de los mismos en caso de que estos tengan lugar.
- **Plan de Negocio** que deberá incluir la información relativa a los criterios de éxito y a la planificación financiera del proyecto software.
- **Plan del Proyecto** que incluirá una planificación general del proyecto software a desarrollar.
- **Plan de Fase** que es una estimación gruesa de la duración y esfuerzo requeridos para la Fase de Elaboración subsiguiente.

---

modelados adecuadamente, es lo que en el contexto de UML se denominan artefactos, siendo `artifacts` el término utilizado en los textos ingleses.

[24] Si el sistema software bajo estudio es una evolución de uno ya existente, o bien está fuertemente vinculado con algún otro sistema ya existente con el que, por ejemplo, debe interactuar, sería necesario un buen conocimiento de dichos sistemas preexistentes.

[25] En documentos posteriores se describen con detalle aquellos entregables de esta fase que se utilizarán en la presente asignatura como herramientas de trabajo. La descripción de otros de los posibles entregables a generar en la Fase de Arranque del Proceso Unificado de Desarrollo de Software como el Plan de Negocio exceden los objetivos de la presente asignatura.

- **Plan de Iteración** que describe qué se hará durante la primera iteración de la Fase de Elaboración subsiguiente.

- **Plan de Desarrollo** que es la propuesta o selección de herramientas de desarrollo, actividades de formación y recursos adicionales que en este sentido pudieran resultar necesarios.

- **Marco de Desarrollo** que es la descripción de los pasos del Proceso Unificado de Desarrollo de Software y de los artefactos UML a generar en cada momento considerados más adecuados para este proyecto software, es decir, la adaptación del Proceso Unificado de Desarrollo de Software para este proyecto software en particular.

- […]

Siguiendo los principios del desarrollo iterativo, no es necesario desarrollar una versión final de todos y cada uno de los documentos anteriores, sino que bastará con realizar el trabajo suficiente como para cumplir el objetivo principal de esta fase que es entender lo suficiente el problema a abordar en el proyecto software como para decidir su viabilidad, es decir, para decidir si el sistema software se puede realizar con la tecnología y recursos disponibles y en el plazo fijado y si podría integrarse con otros sistemas existentes. Debe recordarse además que cualquier documento de los mencionados es opcional y que su utilidad dependerá del problema a abordar. Además, en la práctica siempre es más útil restringirse a unos pocos documentos, los imprescindibles como para poder desarrollar el trabajo en el proyecto software concreto de forma satisfactoria, en vez de dispersar esfuerzos en demasiados documentos que no se piense que resultarán de utilidad.

La duración de esta etapa puede variar enormemente de unos casos a otros ya que el trabajo en esta fase puede consistir en una breve conversación con el cliente, o bien, dicho contacto con el cliente puede prolongarse durante varias semanas e involucrar el uso de diferentes mecanismos como formularios, etc., hasta lograr tener una idea clara de sus necesidades que permita confeccionar la documentación a generar durante esta fase. No obstante, esta fase suele ser corta e, idealmente, no debería alargarse demasiado en el tiempo. En caso contrario, podría existir una excesiva especificación inicial, lo cual iría en contra del enfoque relativamente ágil del Proceso Unificado de Desarrollo de Software.

Durante esta fase, el trabajo estará más centrado en la Captura de Requisitos[26] y en el Análisis que en el resto de Actividades a desarrollar en las iteraciones de las diferentes Fases del Proceso Unificado de Desarrollo de Software: Diseño, Implementación y Pruebas, aunque también se puede abordar alguna de estas actividades, por ejemplo la de Diseño, a través de la realización de prototipos de la Interfaz de Usuario (UI, User Interface) durante esta fase que, típicamente, se incluirán en el Documento de Visión del Sistema.

Fase de Elaboración. Durante esta fase deberían capturarse la mayoría de Requisitos del sistema y elaborarse la mayoría de Casos de Uso. En esta fase, que terminará al alcanzar el hito de la arquitectura inicial del sistema, se realizan típicamente las siguientes tareas:

- Definir de forma precisa los riesgos y las prioridades.

- Capturar la mayoría de los Requisitos.

- Elaborar la mayoría de los Casos de Uso.

[26] En los textos ingleses la Captura de Requisitos se denomina `Requirements Elicitation`. Véase más información en el documento dedicado a este tema de forma específica.

- Crear un plan para la siguiente fase incluyendo asignación de recursos humanos, técnicos, etc.

Al finalizar esta fase se deberá haber alcanzado la comprensión general del proyecto completo. No obstante, en este punto no es necesario alcanzar un conocimiento profundo de todos y cada uno de los aspectos del sistema, ya que dicho conocimiento profundo se adquirirá en fases posteriores y en relación a, únicamente, una parte del sistema en cada iteración.

Fase de Construcción. Es la fase más larga del ciclo de vida. Completa la implementación del sistema tomando como base el resultado obtenido durante la fase de Elaboración. A partir del mismo, las distintas funcionalidades se incluirán en distintas iteraciones, al final de cada una de las cuales se obtendrá una nueva versión ejecutable del producto. Por tanto, esta fase concluye con el hito de obtención de una funcionalidad completa, que capacite al producto para funcionar en un entorno de producción. En esta fase se realizan típicamente las siguientes tareas:

- Completar la captura de Requisitos.
- Completar el Diseño.
- Llevar la Arquitectura a un sistema completo.[27]

Fase de Transición. En la fase final del proyecto se lleva a cabo el despliegue del producto en el entorno de los usuarios, lo que incluye la formación de estos en su uso y quizás en su administración. Por otra parte, en lo relativo al propio producto software:

- Evoluciona desde la fase beta a una versión final gracias a la retroalimentación recogida sobre las versiones preliminares.
- Se trata de resolver incidencias en lo relativo a la integración y a la implantación, y se clasifican y documentan aquellas que podrían justificar una nueva versión del producto.

Esta fase concluye con el hito de lanzamiento o publicación del producto por lo que no debería confundirse con la fase de Integración y Pruebas del Sistema[28] del Modelo Clásico o Modelo en Cascada puesto que al comienzo de la fase de Transición del Proceso Unificado de Desarrollo de Software ya se debería disponer de una versión ejecutable del producto que cubra de forma bastante completa las necesidades del usuario. En esta fase se realizan típicamente las siguientes tareas:

- Preparar la/s máquina/s receptora/s para el sistema software.
- Adaptar el sistema software a la/s máquina/s receptora/s.
- Inserción inicial de datos en caso de ser necesario.[29]
- Crear manuales de usuario.
- Formación del usuario.
- Proporcionar consultoría y soporte al usuario.
- Llevar a cabo la revisión post-proyecto.

---

[27] Entendido como un producto.

[28] La fase denominada `Testing` en los textos ingleses.

[29] Esto podría incluir la conversión de bases de datos a los nuevos formatos, el importar datos preexistentes, etc. Este proceso en los textos ingleses se denomina `data takeon`.

6.5.2 Actividades del Proceso Unificado de Desarrollo de Software

A continuación se describe brevemente cada una de las Actividades que se llevan a cabo en cada una de las fases del Proceso Unificado de Desarrollo de Software:[30]

- **Actividad de Requisitos**: Se trata de descubrir y acordar con el cliente qué debe hacer el sistema. Se deben:
    - **Definir los Casos de Uso**.
    - Producir el **Diagrama de Casos de Uso**[31] inicial.
- **Actividad de Análisis**: Se trata de producir un modelo de análisis. Para ello se deben:
    - **Detallar los Casos de Uso**.
    - Encontrar los conceptos (clases)[32] que intervienen:
        - **Diagrama de Clases**.[33]
        - **Diagrama de Objetos**.[34]
- **Actividad de Diseño**: Se trata de producir un modelo de diseño. Para ello se deben:
    - Diseñar las clases con detalle:
        - **Diagrama de Clases**.
    - Encontrar cómo los objetos de estas clases colaboran entre sí:
        - **Diagrama de Secuencia**.[35]
        - **Diagrama de Colaboración o Comunicación**.[36]
        - **Diagrama de Estados**.[37]
        - **Diagrama de Actividad**.[38]
    - Identificar los subsistemas.
    - Diseñar las interfaces:
        - **Diagrama de Componentes**.[39]

---

[30] En negrita el trabajo a desarrollar vinculado a cada actividad relacionado con el modelado UML.

[31] En los textos ingleses el Diagrama de Casos de Uso se denomina `Use Cases Diagram`.

[32] Los **conceptos** del negocio o del dominio o **clases conceptuales** representados en el **Modelo de Dominio o Modelo Conceptual**, evolucionarán posteriormente a **clases de diseño** en el **Diagrama de Clases del Diseño**. Ambos diagramas, es decir, el Modelo de Dominio o Modelo Conceptual y el Diagrama de Clases del Diseño, se representan haciendo uso del mismo artefacto UML denominado Diagrama de Clases. Véase más información en los documentos dedicados a estos temas de forma específica.

[33] En los textos ingleses el Diagrama de Clases se denomina `Class Diagram`.

[34] En los textos ingleses el Diagrama de Objetos se denomina `Object Diagram`.

[35] En los textos ingleses el Diagrama de Secuencia se denomina `Sequence Diagram`. Los Diagramas de Colaboración o Comunicación y los de Secuencia de UML se denominan **Diagramas de Interacción,** `Interaction Diagrams` en los textos ingleses, y deben ser equivalentes desde un punto de vista semántico. Véase más información en el documento dedicado a este tema de forma específica.

[36] En los textos ingleses el Diagrama de Colaboración o Comunicación se denomina `Collaboration o Communication Diagram`. La denominación preferida en la versión actual de UML es la de Diagrama de Comunicación que remplazaría a la anterior denominación de Diagrama de Colaboración. Los Diagramas de Colaboración o Comunicación y los de Secuencia de UML se denominan **Diagramas de Interacción,** `Interaction Diagrams` en los textos ingleses, y deben ser equivalentes desde un punto de vista semántico. Véase más información en el documento dedicado a este tema de forma específica.

[37] En los textos ingleses el Diagrama de Estados se denomina `State Diagram`.

[38] En los textos ingleses el Diagrama de Actividad se denomina `Activity Diagram`.

[39] En los textos ingleses el Diagrama de Componentes se denomina `Component Diagram`.

    - Diseñar el despliegue:
        - **Diagrama de Despliegue**.[40]

- **Actividad de Implementación**:
    - Implementación de las clases y subsistemas.
    - Pruebas unitarias.
    - Integración.
- **Actividad de Pruebas** de sistema y pruebas en la máquina receptora.

El Proceso Unificado de Desarrollo de Software posee todas las ventajas y desventajas de un Marco de Trabajo Iterativo e Incremental y, por tanto, del Modelo en Espiral del que este deriva. De forma resumida, dichas ventajas y desventajas son las siguientes:

- Ventajas:
    - Retroalimentación temprana y regular del cliente.
    - Testeo temprano del proceso de desarrollo.
    - Prototipos regulares.
    - Tamaño y complejidad del proyecto valorados desde el principio.
    - Riesgos atacados cuanto antes.
- Inconvenientes:
    - Más difícil de entender e interiorizar.
    - Más difícil de gestionar que los procesos lineales.
    - Su adopción supone un cambio muy importante en la forma de trabajar de muchas organizaciones y equipos de desarrollo.

Finalmente deben tenerse muy presente los siguientes errores típicos al hacer uso del Proceso Unificado de Desarrollo de Software para no cometerlos:

- Relacionarlo directamente con el Modelo en Cascada[41]:
    - Inicio ≠ Requisitos
    - Elaboración ≠ Diseño
    - Construcción ≠ Codificación
    - Transición ≠ Integración y Pruebas de Sistema
- Escribir todos los Casos de Uso antes de empezar a programar: La filosofía de trabajo es, por el contrario, el escoger unos pocos Casos de Uso inicialmente, típicamente los más significativos o con un mayor riesgo, para realizar su modelado UML en lo que al análisis y diseño se refiere, y programar su funcionalidad básica, de forma que la integración de las diferentes partes del sistema se vaya realizando de forma paulatina.
- Considerar una duración fija para las iteraciones, de forma, que si no se llega a los plazos de esa iteración se opte por reducir la funcionalidad, es decir, los Casos de Uso, abordados.

[40] En los textos ingleses el Diagrama de Despliegue se denomina `Deployment Diagram`.

[41] Tal y como ya se ha comentado, no debería confundirse, por ejemplo, la fase de Integración y Pruebas del Sistema del Modelo Clásico o Modelo en Cascada con la fase de Transición del Proceso Unificado de Desarrollo de Software puesto que al comienzo de la fase de Transición ya se debería disponer de una versión ejecutable del producto que cubra de forma bastante completa las necesidades del usuario.

7 EL LENGUAJE DE MODELADO UNIFICADO

7.1 ¿ QUÉ ES EL LENGUAJE DE MODELADO UNIFICADO ?

Con anterioridad a UML existían diversos métodos y técnicas de modelado con algunos aspectos en común pero distintas notaciones. La falta de estandarización en la representación visual del modelo dificultaba el aprendizaje e impedía que los diagramas realizados se pudieran compartir fácilmente, además de existir pugnas entre los diferentes enfoques defendidos por los distintos gurús.

El Lenguaje de Modelado Unificado (UML, Unified Modeling Language) pretende dar solución a la situación descrita. El Lenguaje de Modelado Unificado es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad y que cuenta además con el respaldo del OMG (Object Management Group)[42], organización que lo define[43] como un lenguaje de modelado visual que permite especificar, visualizar, construir y documentar un sistema de software proporcionando una abstracción del mismo y de sus componentes, por lo que se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas software a construir.

Es **un lenguaje de modelado y no una metodología** completa. Se puede utilizar, por tanto, para dar soporte a un modelo de proceso de desarrollo de software tal y como lo es el Proceso Unificado de Desarrollo Software, pero no está vinculado a un modelo de proceso de desarrollo de software específico.[44]

También es importante resaltar que el Lenguaje de Modelado Unificado **es un lenguaje de modelado**, o en otras palabras, el lenguaje en el que se describe el modelo. El modelado es la especificación que se hace de un software antes de su codificación, es decir, la visualización de lo que se quiere construir. De igual forma, es importante destacar que el Lenguaje de Modelado Unificado **no es un lenguaje de programación**. No obstante, las herramientas CASE UML pueden ofrecer la funcionalidad de generación de código para diferentes lenguajes de programación, así como la posibilidad de construir modelos por ingeniería inversa a partir de código existente.

El Lenguaje de Modelado Unificado es un lenguaje de propósito general para el modelado y, como cualquier lenguaje de propósito general, pretende ser un lenguaje universal que busca ser tan simple como sea posible, pero manteniendo la capacidad de modelar toda la gama de sistemas software que se necesita construir. Además, necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema software moderno tales como el encapsulamiento.[45]

---

[42] Puede consultarse `http://www.omg.org` para más información. OMG es un consorcio creado en `1989` responsable de la creación, desarrollo y revisión de especificaciones para la industria del software. OMG mantiene la web `http://www.uml.org` sobre el Lenguaje de Modelado Unificado.

[43] Más específicamente, la definición proporcionada por la OMG es la siguiente: UML is a visual language for specifying, constructing and documenting the artifacts of a system.

[44] El hecho de que el Lenguaje de Modelado Unificado no defina un modelo de proceso de software completo puede ser considerado por ciertos desarrolladores como un inconveniente, no así por otros que entienden ambos elementos: Lenguaje de Modelado Unificado y Proceso Unificado de Desarrollo de Software, como herramientas complementarias.

[45] En el contexto del paradigma de programación orientado a objetos, el Encapsulamiento consiste en la propiedad que tienen los objetos de ocultar sus atributos, e incluso sus métodos, a otros objetos con los que interactúan. Por este motivo, la forma natural de construir una clase es la de definir una serie de atributos que, en general, no serán accesibles fuera del propio objeto, sino que únicamente podrán modificarse a través de los métodos del objeto que sean definidos como accesibles desde el exterior de esa clase. Véase más información en el documento dedicado a este tema de forma específica.

El Lenguaje de Modelado Unificado permite definir tanto la estructura estática como el comportamiento dinámico del sistema:

- **Estructura estática**: Cualquier modelo debe definir primero su universo, esto es, los conceptos clave de la aplicación, así como sus propiedades internas. Este conjunto de construcciones es la estructura estática. Los conceptos de la aplicación se modelan como clases,[46] cada una de las cuales se refiere un conjunto de objetos de un determinado tipo que almacenan información modelada como atributos.

- **Comportamiento dinámico**: Existen dos formas de modelar el comportamiento, una es la historia de la vida de un objeto y la forma en que el mismo interactúa con el resto a lo largo del tiempo y la otra es analizando los patrones de comunicación de un conjunto de objetos interconectados, es decir, la forma en que los mismos interactúan entre sí.[47] La visión de un objeto aislado es una máquina de estados, muestra la forma en que el objeto responde a los eventos en función de su estado actual. La visión de la interacción de los objetos se representa con los enlaces entre objetos junto con el flujo de mensajes intercambiados entre los mismos.

El Lenguaje de Modelado Unificado capta la información tanto sobre la estructura estática de un sistema como sobre su comportamiento dinámico y lo modela representándolo a través de una serie de diagramas [ver Figura 13]:

- Diagramas de Estructura:
    - Diagramas de Casos de Uso.
    - Diagramas de Clases.
    - Diagramas de Objetos.
    - Diagramas de Implementación:
        - Diagramas de Componentes.
        - Diagramas de Despliegue.
- Diagramas de Comportamiento:
    - Diagramas de Estados.
    - Diagramas de Actividad.
    - Diagramas de Interacción:
        - Diagramas de Secuencia.
        - Diagramas de Colaboración o Comunicación.

[46] Recuérdese que los conceptos de la aplicación relativos al negocio o dominio de esta se representan inicialmente como clases conceptuales en el Modelo de Dominio o Modelo Conceptual, y posteriormente evolucionan a clases del diseño en el Diagrama de Clases del Diseño.

[47] Es decir, como si se tomara una fotografía del sistema en un instante dado y se analizaran las interacciones entre los objetos que lo componen.

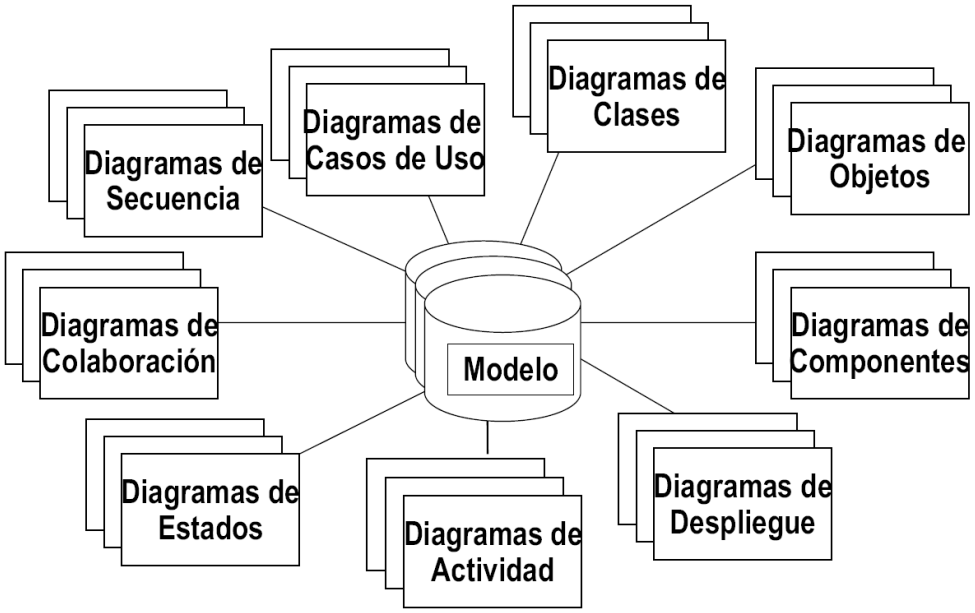


Figura 13: Diagramas de UML.

Finalmente, como inconvenientes del Lenguaje de Modelado Unificado, además del que algunos desarrolladores ven en el hecho de que no defina un proceso completo para el desarrollo de software, se puede señalar que existe una falta de integración con otros elementos como los Patrones de Diseño[48] y el hecho de que es extensible,[49] lo cual hace al Lenguaje de Modelado Unificado muy flexible, pero a riesgo de perder la esencia del estándar como tal.

7.2 HISTORIA DEL LENGUAJE DE MODELADO UNIFICADO

El Lenguaje de Modelado Unificado comenzó a gestarse en Octubre de 1994, cuando Rumbaugh se unió a la compañía Rational Software[50] fundada por Booch[51]. Su objetivo era unificar dos métodos que habían desarrollado: el método Booch y el OMT (Object Modelling Tool). El primer borrador del Lenguaje de Modelado Unificado apareció en Octubre de 1995. En esa misma época otro reputado investigador, Jacobson, se unió a Rational y se incluyeron ideas suyas. Este equipo de trabajo pasó a ser conocido

[48] Los desarrolladores de software que trabajan con el paradigma de Orientación a Objetos con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa tanto el problema como la solución, y se les da un nombre, es a lo que se denomina Patrones de Diseño. Por tanto, puede decirse que un Patrón de Diseño es un par problema/solución que se identifica con un determinado nombre, que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en esas nuevas situaciones y discusiones sobre sus compromisos y contraindicaciones. Los Patrones de Diseño son repetitivos y están probados, y no es, por tanto, su objetivo el expresar nuevas ideas de diseño. Véase más información en el documento dedicado a este tema de forma específica.

[49] Esto supone que es ampliable mediante la incorporación de nuevas características como puedan ser las proporcionadas mediante los denominados Perfiles UML que permiten extender el Lenguaje de Modelado Unificado para así construir modelos UML para dominios particulares.

[50] Rational Software Corporation fue adquirida por IBM en 2003.

[51] Ambos eran por aquel entonces reputados investigadores en el área de la metodología de desarrollo de software.

como los "tres amigos".[52] Al mismo tiempo, el trabajo que se estaba desarrollando se abrió a la colaboración de otras empresas para que aportaran sus ideas.[53] Todas estas colaboraciones condujeron a la definición de la primera versión del Lenguaje de Modelado Unificado.

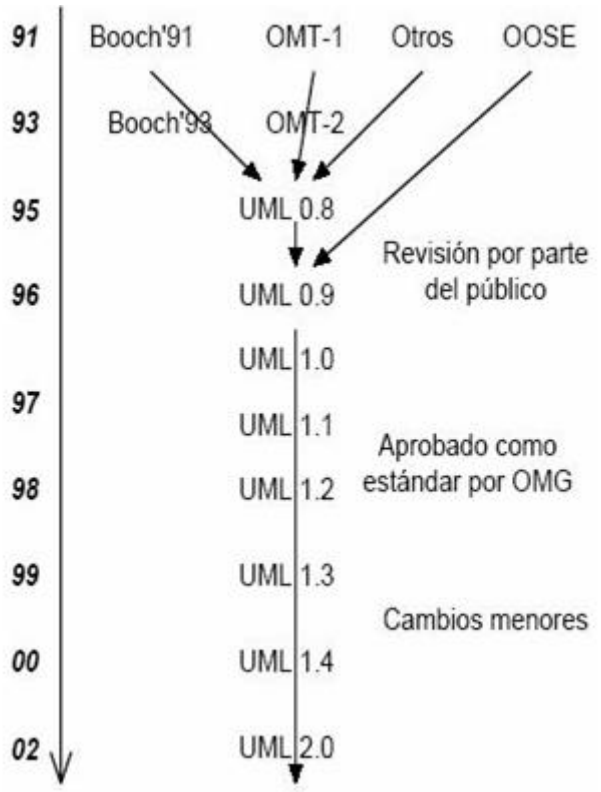


Figura 14: Surgimiento de UML.

Esta primera versión se ofreció a un grupo de trabajo para convertirlo en 1997 en un estándar del OMG. Este grupo de trabajo del OMG que centra su trabajo en estándares relacionados con la tecnología orientada a objetos propuso una serie de modificaciones y una nueva versión del Lenguaje de Modelado Unificado (la 1.1), que fue adoptada por el OMG como estándar en Noviembre de 1997. Desde aquella versión ha habido una serie de revisiones que gestiona la OMG Revision Task Force. La última versión aprobada es la UML 2.0[54] en 2005, aunque ya se está trabajando sobre actualizaciones de esta versión, en concreto UML 2.5[55] se propuso en 2012. Hay que señalar además que la Organización Internacional de Estandarización (ISO, International Standard Organization) aprobó la versión UML 1.0 como estándar "de iure"[56] en 2005. [ver Figura 14]

[52] En los textos ingleses el equipo formado por Rumbaugh, Booch y Jacobson se conoce como The Three Amigos.

[53] Los participantes en UML 1.0 fueron Rational Software (Booch, Rumbaugh y Jacobson), Digital Equipment, Hewlett-Packard, i-Logix, IBM, ICON Computing, Intellicorp and James Martin & co., MCI Systemhouse, Microsoft, ObjecTime, Oracle Corp., Platinium Technology, Sterling Software, Taskon, Texas Instruments y Unisys.

[54] Puede consultarse http://www.omg.org/spec/UML/2.0/ para más información.

[55] Puede consultarse http://www.omg.org/spec/UML/2.5/ para más información.

[56] Se explica el término estándar de iure como la antítesis del término estándar de facto. Un estándar de facto es aquel patrón o norma que se caracteriza por no haber sido consensuada ni legitimada por un organismo de estandarización al efecto. Por el contrario, se trata de una norma generalmente aceptada y ampliamente utilizada por iniciativa propia de un gran número de interesados. Por el contrario, los estándares de iure son aquellos generados por comités con estatus legal y avalados por gobiernos o instituciones como la ISO o la IEEE, y requieren

8 HERRAMIENTAS CASE

8.1 ¿ QUÉ SON LAS HERRAMIENTAS CASE (UML) ?

Como ya es sabido, la Ingeniería de Software necesita de un proceso, de una serie de técnicas con su notación y de un conjunto de herramientas. En concreto, las **herramientas** proporcionan un enfoque automático o semi-automático para el proceso y para las técnicas, estableciéndose un sistema de soporte para el desarrollo del software denominado Ingeniería de Software Asistida por Ordenador (CASE, Computer Aided Software Engineering). Las herramientas son necesarias porque facilitan el seguir el modelo de proceso, así como el utilizar las técnicas y su notación.

Las herramientas CASE son un conjunto de aplicaciones informáticas cuyo objetivo es aumentar la eficiencia en el desarrollo de software, disminuyendo, por tanto, el coste de dicho proceso en términos de recursos invertidos (tiempo, capital humano, etc.) y propiciando una mejora en la calidad del producto software generado. Estas herramientas pretenden servir de ayuda, metódica y automatizada, en diversos aspectos del ciclo de vida del desarrollo del software, en tareas que van desde la planificación o el cálculo de costes hasta la creación de documentación, facilitando además su estandarización, pasando por la generación automática de parte del código. Las herramientas CASE facilitan el uso de la metodología propia de la Ingeniería del Software, además algunas de ellas permiten una gestión integrada de todo el proceso de desarrollo del software, desde el análisis y la captura de requisitos hasta la implementación.

Una herramienta CASE que utilice el Lenguaje de Modelado Unificado como notación para elaborar los modelos – Herramienta CASE UML – permitirá comunicar de forma eficiente a todos los agentes implicados en el proyecto de desarrollo software, todas aquellas decisiones que se vayan tomando en el mismo. Por el contrario, el escoger una herramienta poco adecuada, o bien, un uso poco apropiado de la herramienta escogida puede acarrear un esfuerzo importante en la extracción del conocimiento de los modelos generados.

A través de la notación propuesta por el Lenguaje de Modelado Unificado y con el soporte de una herramienta CASE deberá cumplirse el objetivo de lograr comunicar y compartir de forma eficiente el conocimiento de un determinado sistema software gracias a la combinación simultánea de distintas perspectivas. Así, deberá ser posible definir los diferentes conceptos y trazar sus límites, mantener la trazabilidad entre la concepción de un problema y su solución, usar un vocabulario común o tomar decisiones y evaluar su potencial impacto de manera sencilla y rápida para poder garantizar la coherencia, completitud y usabilidad de los artefactos UML y otros entregables generados.

8.2 CRITERIOS DE SELECCIÓN O DE COMPARACIÓN PARA UNA HERRAMIENTA CASE UML

Algunos de los criterios que se pueden utilizar como criterios de selección o de comparación de herramientas CASE son los siguientes:

- Notación UML:[57]

una elaboración y proceso complejo. Los estándares de facto son la antítesis de los estándares de iure, no obstante, algunos estándares de facto acaban convirtiéndose en estándares de iure.

[57] Clasificación de las herramientas CASE UML propuesta por G. Génova Fuster, J. M. Fuentes Torres y M. C. Valiente Blázquez del departamento de Informática de la Universidad Carlos III de Madrid en su artículo "Evaluación comparativa de herramientas CASE para UML desde el punto de vista notacional".

    - Herramientas gráficas.[58]
    - Herramientas sintácticas.[59]
    - Herramientas semánticas.[60]

- Modelado Visual:
    - Facilidades de dibujo.
    - Convenciones de estilo.
    - […]
- Organización del Repositorio de artefactos UML:
    - Compartir un único repositorio.
    - Control de acceso y versiones para múltiples usuarios.
    - […]
- Diagramas Soportados:
    - Revisión UML.
    - Vista estática.[61]
    - Vista dinámica.[62]
    - […]
- Perfiles – Extensiones UML:[63]
    - Modelado de procesos de negocio.[64]

---

[58] Proporcionan algún tipo de ayuda para dibujar diagramas UML (como plantillas de formas predefinidas, etc.), de modo que resultan algo más convenientes que las herramientas genéricas de dibujo. Sin embargo, no imponen prácticamente ningún tipo de restricción en lo que el usuario puede dibujar, de modo que se facilita la construcción de diagramas UML, pero no su corrección sintáctica. Así, por ejemplo, se podría utilizar un doble subrayado para indicar una instancia nombrada, sin embargo, esto carece de significado en el Lenguaje de Modelado Unificado. Una buena herramienta CASE UML no solo debería ayudar a dibujar diagramas, sino que también debería ayudar a que dichos diagramas fueran correctos.

[59] Facilitan el crear diagramas correctos según las reglas notacionales del Lenguaje de Modelado Unificado. Por tanto, suponen una ayuda mayor para el usuario con respecto a las herramientas meramente gráficas. No obstante, en general estas herramientas no construyen internamente un modelo a la vez que los diagramas que lo expresan, por lo que no permiten comprobar la coherencia entre los distintos diagramas que expresan un mismo modelo desde distintos puntos de vista.

[60] Intentan garantizar la construcción de un modelo que esté correctamente expresado en diagramas que además sean coherentes entre sí.

[61] Compuesta por diferentes diagramas UML como el Diagrama de Casos de Uso o el Diagrama de Clases.

[62] Compuesta por diferentes diagramas UML como el Diagrama de Secuencia o el Diagrama de Colaboración o Comunicación.

[63] Los Perfiles UML son una herramienta que permite extender el Lenguaje de Modelado Unificado y construir así modelos UML para dominios particulares. Un perfil es una colección de extensiones que juntas describen algún problema de modelado en particular, facilitando así la construcción de modelos en ese dominio. Por ejemplo, el perfil UML para XML (Extensible Markup Language) fue definido por David Carlson en su libro "Modeling XML Applications with UML" y el perfil UML para los procesos de negocio fue definido por Hans-Erik Eriksson y Magnus Penker en su libro "Business Modeling with UML. Business Patterns at Work".

[64] Para conseguir sus objetivos, una empresa organiza su actividad por medio de un conjunto de procesos de negocio. Cada uno de ellos se caracteriza por una colección de datos que son producidos y/o manipulados mediante un conjunto de tareas, en las que ciertos agentes (por ejemplo, trabajadores o departamentos) participan de acuerdo con un flujo de trabajo determinado. Además, estos procesos se hallan sujetos a un conjunto de reglas de negocio, que determinan las políticas y la estructura de la información de la empresa. Por tanto, la finalidad del modelado del negocio es describir cada proceso del negocio, especificando sus datos, actividades (o tareas), roles

  - Modelado de datos.[65]
  - Modelado de aplicaciones web (WAE, Web Application Extension).[66]
  - Modelado de la Experiencia del Usuario (UX, User eXperience).[67]
  - Modelado de sistemas de tiempo real.
  - Modelado de DTDs[68] XML[69].
  - […]

- Procesos de Ingeniería [ver Figura 15]:
  - Ingeniería Directa.[70]
  - Ingeniería Inversa.[71]
  - Ingeniería Bidireccional.[72]
  - […]

(o agentes) y reglas de negocio. Recuérdese que, tal y como se ha comentado en la nota al pie anterior, el perfil UML para los procesos de negocio fue definido por Hans-Erik Eriksson y Magnus Penker en su libro "Business Modeling with UML. Business Patterns at Work".

[65] El Diagrama de Clases UML se puede usar para modelar la estructura lógica de una base de datos, independientemente de si es orientada a objetos o relacional, con las clases representando tablas, y los atributos de la clase representando las columnas o campos de la tabla. No obstante, si se escoge la implementación mediante una base de datos relacional que, en el entorno de desarrollo actual, es el método más usado para implementar el almacenamiento de datos, el Diagrama de Clases UML se puede usar para modelar algunos aspectos del diseño de este tipo de bases de datos, pero sin llegar a cubrir toda la semántica involucrada en el modelado relacional (atributos que actúan como claves foráneas para permitir relacionar entre sí unas tablas con otras, etc.). Para capturar esta información se recomienda utilizar el Diagrama Entidad - Relación (ER diagram) como extensión del Lenguaje de Modelado Unificado que es una herramienta de modelado que se concentra en los datos almacenados en el sistema y en las relaciones existentes entre los mismos.

[66] Es una extensión de UML para aplicaciones web creada por Jim Conallen (de Rational Software Corporation, ahora propiedad de IBM). WAE extiende UML para permitir modelar elementos específicos de la Web como parte del modelado de la aplicación, permitiendo modelar, por ejemplo, páginas web, formularios, hiperenlaces o enlaces entre páginas web, applets o pequeños programas desarrollados en el lenguaje de programación Java y embebidos en las páginas web, scripts o códigos JavaScript, etc.

[67] La Experiencia del Usuario, tal y como su propio nombre sugiere, se refiere a la experiencia de los usuarios cuando estos manipulan las interfaces de usuario de las aplicaciones web: aspecto visual de las páginas web, rutas de navegación a través de las páginas web, etc. Jim Conallen en su libro "Building Web Applications with UML" aborda el modelado de UX mediante UML.

[68] Una Definición de Tipo de Documento (DTD, Document Type Definition) es el vínculo entre un Lenguaje de Marcado (ML, Markup Language) y el metalenguaje a partir del cual está definido. El DTD define los elementos y atributos que se podrán utilizar al especificar la estructura de un documento usando dicho Lenguaje de Marcado. Así por ejemplo, el Lenguaje de Marcas de Hipertexto (HTML, Hypertext Markup Language) está basado en el metalenguaje SGML (Standard Generalized Markup Language, Lenguaje de Marcado Estándar Generalizado) y en su DTD se definen sus elementos y atributos como `<p>` – elemento –, `id` – atributo –, etc.

[69] El Lenguaje de Marcado Extensible (XML, Extensible Markup Language) es un metalenguaje más reciente y con una especificación menos voluminosa que SGML (su especificación es un subconjunto de la de SGML) lo cual hace que resulte más sencillo trabajar con él. Existe una reformulación del Lenguaje de Marcado HTML, denominada XHTML (extensible HTML), como una *aplicación* de XML. A los Lenguajes de Marcado en ocasiones se les denominada *aplicaciones*, puesto que se crean *aplicando* las normas de un determinado metalenguaje.

[70] En los textos ingleses a la Ingeniería Directa se le denomina `Forward Engineering`. Es el proceso habitual, en nuestro caso supondría, la transformación de un modelo UML en un código correspondiente en un determinado lenguaje de programación.

[71] En los textos ingleses a la Ingeniería Inversa se le denomina `Reverse Engineering`. De forma amplia, la ingeniería inversa tiene por objetivo el obtener información a partir de un producto final con el fin de determinar de qué está hecho, qué lo hace funcionar o cómo se fabricó. En nuestro caso, supondría la transformación de un código en un determinado lenguaje de programación en su correspondiente modelo UML.

[72] En los textos ingleses a la Ingeniería Bidireccional se le denomina `Round Trip Engineering`. Incluye ambos procesos: el de la Ingeniería Directa y el de la Ingeniería Inversa.

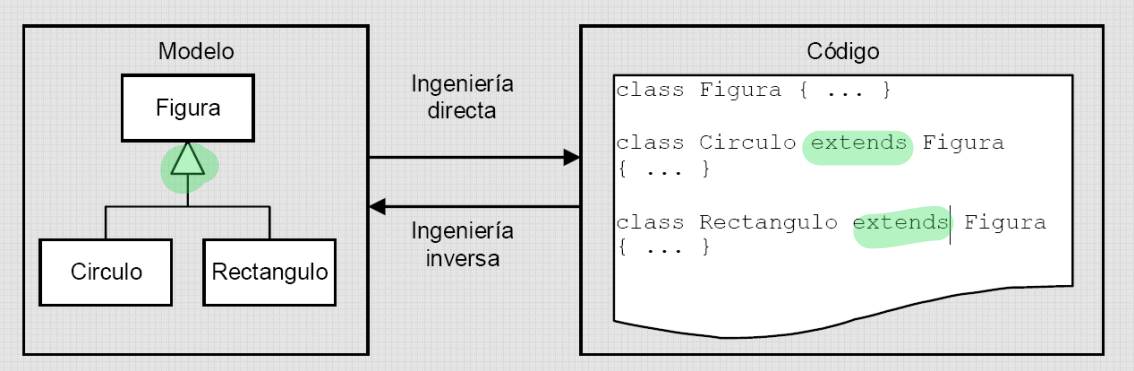


Figura 15: Ingeniería directa, inversa y bidireccional.[73]

- Exportación e Importación de modelos:
  - Lectura y escritura del formato XMI.[74]
  - […]
- Soporte de Arquitectura Dirigida por Modelos (MDA, Model-Driven Architecture)[75]:
  - Especificación de modelos independientes de plataforma.
  - […]
- Soporte de Dirección de Proyectos:
  - Organización de lotes de entregables por iteración.[76]

[73] El esqueleto de código presentado está realizado en el lenguaje de programación Java. Nótese que se definen dos clases, la clase `Circulo` y la clase `Rectangulo`, ambas clases derivadas de la clase base o superclase `Figura` tal y como se ha especificado en el modelo.

[74] XMI (XML Metadata Interchange, XML para Intercambio de Metadatos) es un estándar que permite expresar en forma de fichero XML cualquier modelo (o meta-modelo) definido en MOF (Meta Object Facility). De esta forma XMI ofrece un formato adecuado para el intercambio entre diferentes herramientas de aquella información cuya semántica se haya expresado en MOF. XMI permite, por tanto, compartir modelos UML entre diferentes herramientas de modelado que soporten este estándar. XMI, al igual que MOF, ha sido creado por el OMG (Object Management Group, `http://www.omg.org/index.htm`) que es un consorcio responsable de la creación, desarrollo y revisión de especificaciones para la industria del software. Puede consultarse `http://www.omg.org/technology/xml/index.htm` para más información.

[75] El OMG ha creado un marco conceptual para separar las decisiones orientadas-al-negocio de aquellas orientadas-a-la-plataforma-de-desarrollo. Este marco y los estándares que ayudan a implementarlo es lo que OMG denomina MDA. MDA separa, por tanto, la lógica de negocio de la aplicación, de la plataforma tecnológica subyacente, permitiendo la realización de modelos para aplicaciones independientes de la plataforma tecnológica que se vaya a utilizar. Estos modelos independientes de la plataforma documentan el funcionamiento del negocio de una aplicación separándolo de la tecnología específica que lo implementará, aislando así el núcleo de la aplicación de la tecnología. Así, al no estar ligadas ambas componentes (negocio y tecnología) pueden cada una evolucionar al ritmo que se considere necesario. Puede consultarse `http://www.omg.org/mda/` para más información.

[76] Hace referencia a las iteraciones del Proceso Unificado de Desarrollo de Software, que es el modelo de proceso o ciclo de vida del software o paradigma de Ingeniería de Software más frecuentemente utilizado en la actualidad. Como ya es sabido, el Proceso Unificado de Desarrollo de Software es un marco de desarrollo iterativo e incremental. Esto supone que es un marco de trabajo que propone múltiples ciclos cada uno comprendiendo las fases de Arranque o Inicio, Elaboración, Construcción y Transición. En este marco de desarrollo, cada una de las fases anteriores puede incluir a su vez una serie de *iteraciones* que ofrecen como resultado un incremento del producto desarrollado, que añade o mejora las funcionalidades del sistema en desarrollo, aunque normalmente, en

  - Definición de roles participantes en la elaboración de los artefactos de modelado.
  - Imputación de tiempo en la elaboración de artefactos de modelado.
  - Previsión y métrica de los riesgos del proyecto.
  - […]
- Soporte del Ciclo de Vida de un Proyecto:
  - Integración con herramientas de trazabilidad de requisitos.[77]
  - Integración con herramientas de control de versiones.
  - Integración con analizadores de carga y rendimiento.
  - […]
- Soporte de la Implementación:
  - Integración de diferentes generadores de código.
  - […]
- Generación de Informes y Documentación:
  - Integración con generadores de informes, diccionarios de datos, cuadernos de proyecto, etc.
  - Vincular artefactos de modelado con documentos de especificación.
  - Informes de validación de sintaxis y reglas de notación UML.
  - Importación de diagramas para confeccionar presentaciones.
  - […]
- Publicación web:
  - Generación de documentación en (X)HTML.
  - […]
- Soporte:
  - Aprendizaje escalable.
  - Muestrario de casos.
  - Tutoriales de entrenamiento.
  - Servicios de tutelaje para desarrollar proyectos piloto.
  - […]
- Escalabilidad de Costes y Prestaciones:
  - Proyecto de código abierto.
  - Política de licencias para equipos de proyecto.

---

la práctica, solo se planifican iteraciones para la fase de Construcción. Durante cada una de estas iteraciones se realizarán a su vez una serie de posibles actividades: Requisitos, Análisis, Diseño, Implementación y Pruebas (e Implantación). Aunque todas las iteraciones pueden incluir trabajo en todas estas actividades, el grado de esfuerzo dentro de cada una de ellas varía a lo largo del proyecto. Así, por ejemplo, la fase de Arranque o Inicio se centrará más en la definición de Requisitos y en el Análisis, actividades que durante la fase de Construcción quedarán relegadas en favor de las actividades de Implementación y Pruebas.

[77] La trazabilidad de requisitos se define como la habilidad para describir y seguir la vida de un Requisito en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de todas las especificaciones generadas durante el proceso de desarrollo de software.

- Disponibilidad de versiones gratuitas para fines educativos.
- […]

8.3 EJEMPLOS DE HERRAMIENTAS CASE UML

A continuación, se proporciona una lista de diferentes herramientas CASE UML. No todas ellas proporcionan una funcionalidad comparable. Así, por ejemplo, en la citada lista hay herramientas CASE UML que proporcionan una funcionalidad bastante completa como ArgoUML frente a otras más limitadas y probablemente sencillas como Violet, o incluso algunas como Use Case Maker que se centran en un aspecto concreto, en este caso en el Modelo de Casos de Uso. Algunas de ellas son comerciales como Rational y otras son proyectos de código abierto como ArgoUML, también varias de ellas, aunque no son gratuitas, cuentan con una versión básica que sí es gratuita o tiene un precio más asequible. Algunas de las herramientas CASE UML propuestas están expresamente diseñadas para algún Entorno Integrado de Desarrollo (IDE, (Integrated Development Enviroment)) como por ejemplo NetBeans IDE UML o Eclipse UML2 Tools. La mayoría son aplicaciones independientes que es posible instalar en la máquina en la que se trabaje, no obstante, otros son servicios disponibles en línea, en concreto, WebSequenceDiagram o yUML.

Lista de herramientas CASE UML:

- Apollo
- ArgoUML[78]
- BoUML[79]
- Dia[80]
- Eclipse Omondo[81]
- Eclipse UML2 Tools[82]
- Enterprise Arquitect[83]
- NetBeans IDE UML[84]
- Pacestar UML Diagrammer[85]
- Poseidon
- Rational
- StarUML[86]

---

[78] Puede consultarse `https://argouml.uptodown.com/windows` para más información.

[79] Puede consultarse `http://www.bouml.fr/` para más información.

[80] Puede consultarse `https://wiki.gnome.org/Apps/Dia` para más información.

[81] Puede consultarse `http://www.omondo.com/` para más información.

[82] Puede consultarse `http://wiki.eclipse.org/MDT-UML2Tools`.

[83] Puede consultarse `http://www.sparxsystems.com/products/ea/index.html` para más información.

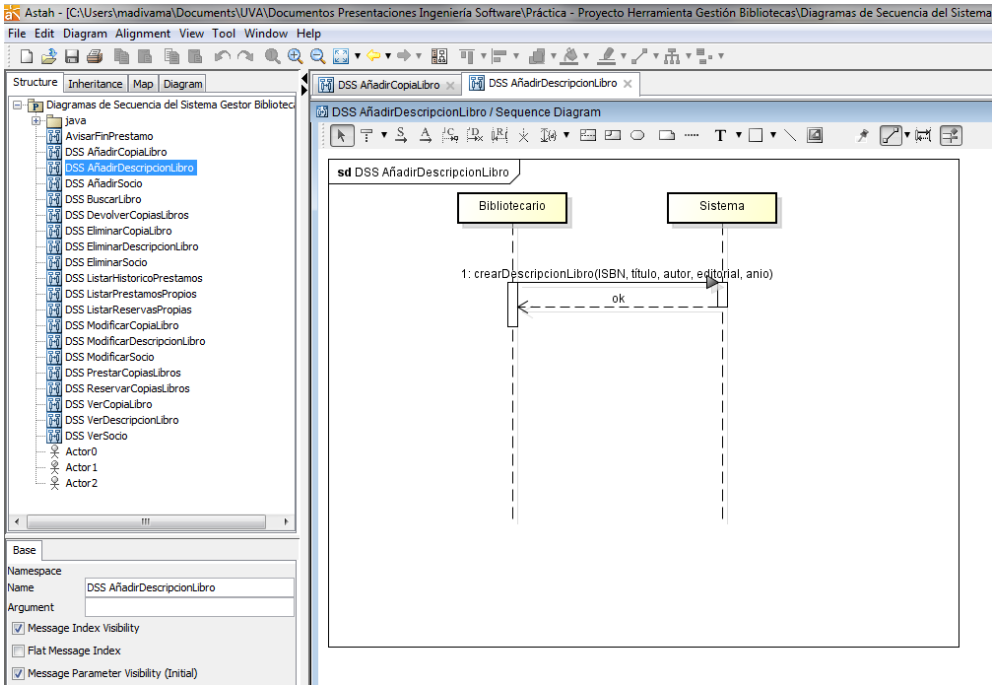
[84] Puede consultarse `http://netbeans.org/features/uml/` para más información.

[85] Puede consultarse `http://www.pacestar.com/uml/index.html` para más información.

- T4 Editor plus UML modeling Tools for Visual Studio[87]
- Umbrello UML Modeller[88]
- UMLet[89]
- Use Case Maker[90]
- Violet UML Editor[91]
- WebSequenceDiagram[92]
- yUML[93]

8.4 ASTAH

Como herramienta CASE UML para trabajar en el laboratorio durante el presente curso académico se ha escogido Astah [ver Figura 16].[94]



[86] Puede consultarse `http://staruml.io/` para más información.

[87] Puede consultarse `http://t4-editor.tangible-engineering.com/T4-Editor-Visual-T4-Editing.html` para más información.

[88] Puede consultarse `https://umbrello.kde.org/` para más información.

[89] Puede consultarse `http://www.umlet.com/` para más información.

[90] Puede consultarse `http://use-case-maker.sourceforge.net/` para más información.

[91] Puede consultarse `http://alexdp.free.fr/violetumleditor/page.php` para más información.

[92] Puede consultarse `http://www.websequencediagrams.com/` para más información.

[93] Puede consultarse `http://yuml.me/diagram/scruffy/class/draw` para más información.

[94] Puede consultarse `http:// http://astah.net/` para más información.

Figura 16: Vista de la herramienta CASE Astah.

9 EN ESTA ASIGNATURA…

Finalmente, y volviendo al planteamiento realizado en la Figura 4: ¿Qué necesita la Ingeniería de Software?, señalar que, en esta asignatura

- El proceso utilizado será el Proceso Unificado de Desarrollo de Software.
- La notación a utilizar será el Lenguaje de Modelado Unificado.
- Y la herramienta CASE UML a utilizar será Astah.

10 BIBLIOGRAFÍA

Ariadne-1. (2001). *UML Applied – Object Oriented Analysis and Design. First Edition.* Ariadne Training.

Ariadne-2. (2005). *UML Applied – Object Oriented Analysis and Design. Second Edition.* Ariadne Training.

Bauer, F. L. (1972). *Software engineering, Information Processing.* Amsterdam: North-Holland Publishing Co.

Bohem, B. (1976). Software Engineering. *IEEE Transactions on Computers 25 (12)* , 1226-1241.

Pressman, R. S. (2002). *Ingeniería de Software. Un Enfoque Práctico (quinta edición ed.).* Madrid: Mc Graw Hill.

Royce, W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. *Technical Papers of Western Electronic Show and Convention (WesCon).* Los Angeles, USA.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *El Proceso Unificado de Desarrollo de Software.* Addison Wesley.

Sommerville, I. (2005). *Ingenieria de Software.* Pearson Education.

Zelkovitz, M. V. (1978). *Principles of Software Engineering and Design.* Prentice Hall.