

Dr. Francisco José García Peñalvo

Ciencia de la Computación e Inteligencia Artificial

Universidad de Salamanca

Capítulo 7. Ingeniería del Software

One principle problem of educating software engineers is that they will not use a new method until they believe it works and, more importantly, that they will not believe the method will work until they see it for themselves

Watts S. Humphrey

La incapacidad de las organizaciones para predecir tiempo, esfuerzos y costes en el desarrollo de *software* producido son dos de las principales bases sobre las que surge la *Ingeniería del Software* como una disciplina científica.

Otros factores que contribuyen a la creación de esta disciplina son:

- *Cambios en la relación de costes hardware/software.* Durante los primeros años de desarrollo de las computadoras, el *hardware* sufrió continuos cambios, mientras que el *software* no era considerado sino como un añadido a una máquina, el coste de la cual, además, absorbía la mayor parte del presupuesto destinado a la adquisición del sistema informático. Por ello, en la mayoría de los casos los programas se construían con la finalidad de hacer el uso de las máquinas eficiente, pero en absoluto se pensaba en un desarrollo eficiente de los mismos. Sin embargo, la situación fue cambiando y ya a mediados de los años 60 los costes del *software* ascendieron hasta un 40-50% del coste total del sistema, y su influencia fue creciendo hasta niveles en los que el coste del *hardware* ya representaba tan solo el 20% del total.
- *La importancia creciente del mantenimiento.* La tarea de un desarrollador de *software* no concluye cuando el producto es implementado, instalado y entregado. En efecto, cuando comienza la explotación de un producto *software* se detectan errores, los usuarios en ocasiones piden cambios del mismo, los propios productores pueden facilitar nuevas versiones mejoradas, etc. Todo ello suele considerarse como mantenimiento del *software*. Las tareas de mantenimiento pueden llegar a ser más dificultosas (y, en consecuencia, más caras) que las de desarrollo de un nuevo producto.
- *Los avances en el desarrollo del hardware.* Conforme las generaciones de computadoras se iban sucediendo, los costes de producción de las mismas disminuyeron considerablemente. Esto trajo consigo la implantación casi generalizada del ordenador como herramienta habitual de trabajo, ya que tanto grandes como pequeñas empresas tenían a su alcance algún modelo que cumplía con sus necesidades de proceso de datos a un precio asequible. Evidentemente, el aumento en las ventas de computadores trajo consigo un crecimiento enorme de la demanda de nuevos programas que cubriesen todas las necesidades de los potenciales usuarios.
- *Demanda de software más complejo.* Los avances tecnológicos han propiciado igualmente la aparición de grandes sistemas *software* más complejos que los que pudieran existir en el pasado que resuelven más problemas que los sistemas pequeños. A medida que dichos grandes sistemas se fueron desarrollando, se hizo patente que las técnicas empleadas para desarrollar pequeños productos (si es que dichas técnicas existían, ya que muchas veces la programación ha sido

considerada más arte que ciencia) no funcionaban para aquellos. La necesidad de una aproximación disciplinada al desarrollo de grandes y/o complejos sistemas *software* era, por tanto, evidente. El incremento de la complejidad del *software* no solo se debe al tamaño de los sistemas, también va a influir el aumento de la facilidad de uso de los sistemas *software* y el aumento del número de conexiones en red.

Con el objetivo de vencer todas estas dificultades surgió una nueva disciplina conocida como *Ingeniería de Software*, cuyo nombre se propuso en 1968 en una conferencia de la Organización del Tratado del Atlántico Norte (OTAN) [1] para analizar los problemas del desarrollo de *software*; en esa época había grandes sistemas de *software* que estaban rezagados, que no ofrecían la funcionalidad que requerían los usuarios, que costaban más de lo esperado y que no eran fiables [2]. Como primera aproximación podría tomarse a definición del ANSI/ IEEE Std 729-1983:

The systematic approach to the development, operation, maintenance, and retirement of software (La aproximación sistemática al desarrollo, operación, mantenimiento y retirada del software) [3] (p. 32).

Según lo anterior, el desarrollo del *software* es un problema ingenieril ya que trata de crear soluciones efectivas y viables económicamente hablando a problemas reales.

Antes de entrar a una mayor conceptualización de la *Ingeniería del Software*, conviene dejar completamente clara la definición de *software*, por ejemplo, según el IEEE Std 610.12-1990, se puede definir como:

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system [4] (p. 66).

O como lo definen Roger S. Pressman y Bruce R. Maxim:

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information, and (3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs [5] (p. 4).

Por su parte Mario G. Piattini y otros definen *software* como:

Conjunto de programas, procedimientos y documentación asociada a la operación de un sistema informático [6, 7].

Los documentos son en la mayoría de los casos tan importantes para el correcto aprovechamiento del *software* como lo es el propio programa fuente. De hecho, hay estimaciones que sitúan al desarrollo de la documentación asociada a los sistemas *software* entre los dos o tres apartados más costosos de todo el proceso [8].

7.1. Definición de Ingeniería del Software

La introducción del término *Ingeniería del Software* se produce en la primera conferencia sobre Ingeniería del Software patrocinada por la OTAN, celebrada en Garmisch (Alemania) en octubre de 1968 [1], no obstante la paternidad del término se le atribuye a Fritz Bauer [9].

Fue tal la aceptación de esta conferencia que se consiguió un nuevo patrocinio de la OTAN para una segunda conferencia sobre Ingeniería del Software, que tendría lugar un año más tarde en Roma (Italia) [10], con unos resultados menos esperanzadores que los producidos en la primera conferencia. De hecho, no se produjo ninguna petición de que continuara la serie de conferencias de la OTAN, lo cual no influyó para que a partir de entonces se utilizara con gran profusión el nuevo término para describir los trabajos realizados, aunque quizás sin un consenso real sobre su significado.

En el contexto educativo, sin duda alguna, lo que más controversia ha levantado es el propio nombre del término, centrándose la discusión en la pregunta *¿es la Ingeniería del Software realmente una Ingeniería?* [11-13].

Los argumentos que se dan para sustentar una respuesta negativa se pueden resumir en dos categorías. La primera reuniría a aquellos que se ciñen a la definición literal de ingeniería dada por algunos diccionarios o sociedades profesionales, argumentando que en estas definiciones se hace mención a productos tangibles derivados del uso efectivo de materiales y fuerzas naturales, mientras que el *software* ni es tangible, ni utiliza materiales y/o fuerzas naturales para su concepción. La segunda categoría estaría formada por los que arguyen que una disciplina ingenieril evoluciona desde una profesión y la profesión relacionada con el *software* no ha evolucionado lo suficiente para ser considerada una ingeniería.

Por el contrario, son muchos los que están a favor de la utilización y difusión del término *Ingeniería del Software*, tomando como un estándar de facto la utilización reiterada del término en la bibliografía especializada.

Quizás la defensa más fuerte y adecuada de la Ingeniería del Software como ingeniería venga de la mano de Mary Shaw que justifica que si tradicionalmente se ha definido ingeniería como “la creación de soluciones rentables a problemas prácticos mediante la aplicación del conocimiento científico para la construcción de cosas al servicio de la humanidad” [14], entonces el desarrollo del *software* es un problema ingenieril apropiado, porque involucra “la creación de soluciones rentables económicamente para problemas prácticos” [15].

Una vez hechas estas disquisiciones sobre el término y sus controversias, se va a proceder a exponer una muestra de las numerosas definiciones que de Ingeniería del Software se pueden encontrar en la bibliografía.

Ingeniería del software es el establecimiento y uso de principios sólidos de ingeniería, orientados a obtener software económico que sea fiable y trabaje de manera eficiente en máquinas reales. Fritz Bauer, First NATO Software Engineering Conference, Garmisch (Germany), 1968 [16, 17].

Multi-person development of multi-version programs. David L. Parnas [18, 19].

Software Engineering: The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them. Barry W. Boehm [20] (p. 1226).

Ingeniería del Software es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas software. M. V. Zelkowitz et al. [21].

Es la disciplina tecnológica y de gestión que concierne a la producción y mantenimiento sistemático de productos software que son desarrollados y modificados a tiempo y dentro de los costes estimados. Richard Fairley [22].

Tratamiento sistemático de todas las fases del ciclo de vida del software. Se refiere a la aplicación de metodologías para el desarrollo del sistema software. Asociación Española para la Calidad [23].

La aplicación disciplinada de principios, métodos y herramientas de ingeniería, ciencia y matemáticas para la producción económica de software de calidad. Watts S. Humphrey [24].

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1).

IEEE Std 610.12-1990 [4] (p. 67).

(1) The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.

(2) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

24765-2010 - ISO/IEC/IEEE [25] (p. 331).

Disciplina tecnológica y de gestión concerniente a la invención, producción sistemática y mantenimiento de productos software de alta calidad, desarrollados a tiempo y al mínimo coste. William B. Frakes et al. [26].

Aplicación de herramientas, métodos y disciplinas para producir y mantener una solución automatizada de un problema real. Bruce I. Blum [27].

Aplicación de principios científicos para la transformación ordenada de un problema en una solución software funcional, así como en el consiguiente mantenimiento del software hasta el final de su vida útil.

Alan M. Davis [28].

That form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problem. **Watts S. Humphrey** [29].

La aplicación de métodos y conocimiento científico para crear soluciones prácticas y rentables para el diseño, construcción, operación y mantenimiento del software y los productos asociados, al servicio de las personas. **Mary Shaw y David Garlan** [30].

The application of science and mathematics by which the properties of software are made useful to people. **Barry W. Boehm** [31] (p. 12).

Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high-quality computer software. **Roger S. Pressman y B. R. Maxim** [5] (p. 14).

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

Ian Sommerville [32] (p. 21).

La Ingeniería del Software requiere la comprensión y aplicación de principios de ingeniería, habilidades de diseño, buenas prácticas de gestión, fundamentos de la Ciencia de la Computación y formalismos matemáticos. Es tarea de la Ingeniería del Software juntar estas áreas de trabajo tan dispares y utilizarlas en las fases de obtención de los requisitos, especificación, diseño, verificación, implementación, prueba,

documentación y mantenimiento de sistemas software complejos y de gran tamaño. El ingeniero del software debe cumplir el papel del arquitecto del sistema complejo, tomando en cuenta las necesidades y requisitos del usuario, la viabilidad, el coste, la calidad, la confianza, la seguridad y las restricciones temporales. La necesidad de ajustar la importancia relativa de estos factores de acuerdo a la naturaleza del sistema y de su aplicación confiere una fuerte dimensión ética a las tareas del ingeniero del software, sobre quien pueda depender la seguridad y bienestar de otros, y para quien, como en medicina o en derecho, un sentido de moralidad profesional se requiere para su trabajo.

El ingeniero del software debe ser capaz de estimar el coste y la duración del proceso de desarrollo del software, así como determinar la consecución de corrección y confianza. Tales medidas y estimaciones pueden involucrar conocimientos de conceptos financieros y de gestión, al mismo nivel que el manejo de los fundamentos matemáticos. Se necesita el uso preciso de las notaciones formales y de las palabras para expresarlas con el grado de precisión requerido a otros ingenieros y a clientes formados. En la mayoría de las circunstancias las hebras técnicas, teóricas y de gestión de un proyecto de Ingeniería del Software no pueden separarse unas de las otras.

Para construir grandes productos y conseguir una alta productividad, el ingeniero requiere el uso de herramientas software de desarrollo y de elementos reutilizables que garanticen su subsiguiente modificación y mantenimiento con seguridad.

La actividad profesional del ingeniero del software abarca el rango de tareas involucradas en el ciclo de vida de un sistema software. La obtención de requisitos, especificación, diseño, verificación y construcción son tareas críticas para conseguir la calidad del producto y son todas ellas responsabilidad del ingeniero del software.

Dado que el software determina el comportamiento de un autómeta, el ingeniero del software necesita tener conocimientos de hardware digital y de comunicaciones. Aunque la Ingeniería del Software como disciplina puede ser calificada como independiente del área de aplicación, su realización debe ser en el contexto de aplicaciones específicas. El ingeniero del software debe, por tanto, ser capaz de colaborar con otros profesionales que le brindarán capacidades complementarias en la labor de especificar,

diseñar y construir sistemas hardware-software que se ajusten a las necesidades del cliente, haga uso de las soluciones hardware y software en una óptima combinación y ofrezca una interfaz de usuario con una calidad adecuada.

La mayoría del software se construye en equipo, frecuentemente con equipos interdisciplinarios. La habilidad para trabajar cerca los unos de los otros es esencial.

*Algunos de los métodos y de las herramientas intelectuales de la Ingeniería del Software están en proceso de desarrollo y se espera que tengan que cambiar de forma rápida. Los ingenieros del software, por tanto, necesitan tener unos buenos fundamentos teóricos que les sirvan de base para aprender y usar nuevos métodos en el futuro, y la mentalidad que les permita actualizar de forma permanente los conocimientos que necesitan para su labor profesional. **The British Computer Society and The Institution of Electrical Engineering** [33].*

1. *Definición central:*

- *Ingeniería es la aplicación sistemática de conocimiento científico para la creación y construcción de soluciones rentables a problemas prácticos al servicio de la humanidad.*
- *La Ingeniería del Software es la forma de ingeniería que aplica principios propios de la Ciencia de la Computación y Matemáticas para conseguir soluciones rentables a problemas software.*

2. *Elaboraciones e interpretaciones:*

- *La creación y construcción del software debe incluir el mantenimiento. Debe cubrirse el ciclo de vida del software completo.*
- *La rentabilidad implica no solo dinero, sino tiempo, calendario y recursos humanos. También implica obtener buenos valores por los recursos invertidos; lo que incluye la calidad cuando las medidas se consideren oportunas.*
- *La Ingeniería del Software no se limita a aplicar solo principios de la Ciencia de la Computación y las Matemáticas, sino cualquier principio del que pueda sacar ventaja.*

- *La Ingeniería del Software necesita contar con principios y técnicas de gestión para llevar a cabo sus actividades de desarrollo.*
3. *Distinción entre el uso actual del término “Ingeniería del Software” y la definición que se adecua a la misión del Software Engineering Institute:*
- *Actualmente, el término “Ingeniería del Software” tiene múltiples conjuntos de significados conflictivos y pobremente entendidos, que van desde la programación a la gestión del diseño del sistema.*
 - *Actualmente, el término “Ingeniería del Software” es más una aspiración que una descripción.*

Software Engineering Institute [34].

Parece claro que hay un consenso en que el desarrollo del *software* necesita una base rigurosa que encuentra en la Ingeniería, e indirectamente en la Ciencia y en las Matemáticas. Pero concretamente, en lo referente a la palabra *Ingeniería*, hay diferentes opiniones, pero con el paso del tiempo tienden a consensuarse en que la *Ingeniería del Software* es una disciplina de *Ingeniería*. Por ejemplo, Hoare [35], en la década de los setenta, cita los componentes clave de la ingeniería, que según él son lo suficientemente valiosos y relevantes como para ser imitados por los desarrolladores de *software*, concretamente identifica cuatro aspectos de la ingeniería que cubren tanto los elementos teóricos como los prácticos de la Ingeniería del Software: *profesionalismo, vigilancia, conocimiento teórico y herramientas*; J. A. McDermid [36] destaca los fundamentos de Ciencia y Matemáticas; R. S. Pressman, en ediciones anteriores y en clara alusión a F. Bauer, habla simplemente de principios de ingeniería, que pueden incorporar principios teóricos y prácticos [37], sin embargo, en su última edición pone mucho más énfasis en el proceso y en los métodos, es decir, en los aspectos prácticos, para lograr el fin último, el *software* de calidad, para lo que se presenta la Ingeniería de Software organizada en capas y sustentada en el proceso (ver Figura 7.1), el cual si bien requiere disciplina, necesita también de adaptabilidad y de agilidad [5]; algo similar ocurre con Ian Sommerville que, por ejemplo en la edición de su libro incluye “teorías, métodos y herramientas” [38], aunque indica que la Ingeniería del Software es diferente a otras formas de Ingeniería debido a la propia naturaleza del *software*, sin embargo, en la última edición comienza su definición explícitamente con “La Ingeniería del Software es una disciplina ingenieril” y pone mucho énfasis en que la Ingeniería del Software no

solo se refiere a los procesos técnicos para el desarrollo del *software*, sino que además incluye las actividades de gestión del proyecto, así como el desarrollo de herramientas, métodos y teorías para el soporte del desarrollo del *software*; Anthony I. Wasserman sugiere que existen ocho nociones fundamentales en la Ingeniería del Software que forman la base para una disciplina efectiva [39]: abstracción, métodos y notaciones de análisis y diseño, prototipado de la interfaz de usuario, modularidad y arquitectura del *software*, proceso y ciclo de vida, reutilización, métricas, y herramientas y entornos integrados.

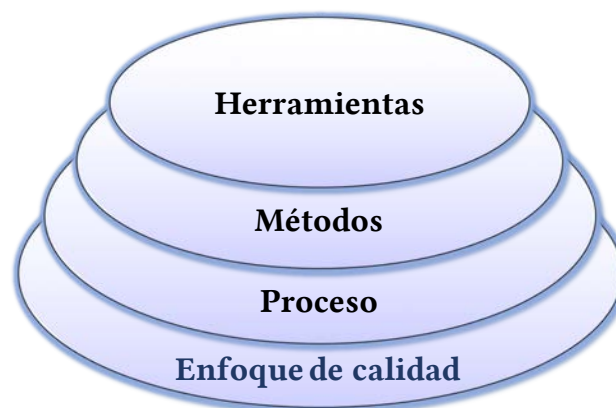


Figura 7.1. Capas de la Ingeniería del Software. Fuente: Basado en [5] (p. 16)

No solo hace falta decir lo qué es la Ingeniería del Software, sino que es conveniente recalcar lo qué no es; así la Ingeniería del Software no es el diseño de programas que se implementan en otras áreas ingenieriles, ni es simplemente una forma de programar más organizada que la que prevalece entre aficionados, principiantes o personas con falta de educación y entrenamiento específico.

El concepto de Ingeniería del Software surge de la distinción entre el desarrollo de pequeños proyectos (*programming in the small*) y el desarrollo de grandes proyectos (*programming in the large*), de forma que el reconocimiento de que la Ingeniería del Software está relacionada con esta última. Este primer concepto fue rápidamente ampliado para incorporar a la Ingeniería del Software todas aquellas tareas relacionadas con la automatización de los Sistemas de Información y con la Ingeniería de Sistemas en general.

7.2. Marco conceptual de la Ingeniería del Software

El estudio de las características comunes de los sistemas se conoce como Teoría General de Sistemas [40]. Los principios de esta teoría, derivados del estudio de otros sistemas,

se pueden aplicar a los sistemas automatizados. Algunos de los principios generales de la teoría general de sistemas son los siguientes:

- Cuanto más especializado sea un sistema menos capaz es de adaptarse a circunstancias diferentes.
- Cuanto mayor sea el sistema mayor es el número de recursos que deben dedicarse a su mantenimiento diario.
- Los sistemas siempre forman parte de sistemas mayores y siempre pueden dividirse en sistemas menores.
- Los sistemas crecen.

La Ingeniería de Sistemas es uno de los antecedentes de la Ingeniería del Software. En ella las funciones deseadas de un sistema se descubren, analizan y se asignan a elementos de sistemas individuales. El ingeniero de sistemas parte de los objetivos definidos por el usuario y desarrolla una representación de la función, rendimiento, interfaces, restricciones de diseño y estructura de la información que pueden ser asociados a cada uno de los elementos del sistema (documentos, procedimientos, bases de datos, etc.).

En el marco de la Teoría General de Sistemas, el análisis de sistemas tiene como objetivo general la comprensión de los sistemas complejos para abordar su modificación de forma que se mejore el funcionamiento interno para hacerlo más eficiente, para modificar sus metas, etc. Las modificaciones pueden consistir en el desarrollo de un subsistema nuevo, en la agregación de nuevos componentes, en la incorporación de nuevas transformaciones, etc. En general, el análisis de sistemas establece los siguientes pasos a seguir:

1. *Definición del problema.* En este paso se identifican los elementos de insatisfacción, los posibles cambios en las entradas y/o salidas al sistema y los objetivos del análisis del sistema.
2. *Comprensión y definición del sistema.* En este paso se identifica y descompone el sistema jerárquicamente en sus partes constituyentes o subsistemas junto con las relaciones existentes entre los mismos.
3. *Elaboración de alternativas.* En este paso se estudian las diferentes alternativas existentes para la modificación y mejora del sistema, atendiendo a los costes y perspectivas de realización.

4. *Elección de una de las alternativas definidas en el paso anterior.*
5. *Puesta en práctica de la solución elegida.*
6. *Evaluación del impacto de los cambios introducidos en el sistema.*

Desde una perspectiva más general, Jean-Louis Le Moigne [41] concibe los sistemas formados por tres subsistemas interrelacionados: el de decisión, el de información y el físico. El sistema de decisión procede a la regulación y control del sistema físico con el objetivo de definir su comportamiento en función de los objetivos marcados. El sistema físico transforma un flujo físico de entradas en un flujo físico de salidas. En interconexión entre el sistema físico y el sistema de gestión se encuentra el sistema de información. El sistema de información está compuesto por diversos elementos encargados de almacenar y tratar las informaciones relativas al sistema físico a fin de ponerlas a disposición del sistema de gestión. El Sistema Automatizado de Información (SAI) es un subsistema del sistema de información en el que todas las transformaciones significativas de información son efectuadas por máquinas de tratamiento automático de las informaciones.

Con base en estas ideas, se considera a la Ingeniería del Software como la disciplina que se ocupa de las actividades relacionadas con los sistemas informáticos o sistemas de información en los que el *software* desempeña un papel relevante. Estos sistemas de información han de ser fiables, es decir, que su realización se lleve a cabo de forma correcta conforme a unos estándares de calidad y, además, que su desarrollo se realice en el tiempo y coste establecidos.

No obstante, la Ingeniería del Software va más allá. Abarca un conjunto de tres elementos clave: métodos, herramientas y procedimientos, que facilitan al gestor controlar el proceso de desarrollo del *software* y suministrar a los que practiquen dicha ingeniería las bases para construir *software* de alta calidad de una forma productiva.

El término *software* de calidad aparece reiteradamente como un fin de la Ingeniería del Software. Una valoración general de la calidad de un sistema *software* requiere la identificación de atributos comunes que pueden esperarse de un buen producto de ingeniería. Aunque se asuma que el *software* proporciona la funcionalidad requerida, hay una serie de atributos que se deberían encontrar, porque además la calidad subjetiva de un sistema *software* está mayormente relacionada con sus atributos no funcionales (seguridad, protección, fiabilidad, flexibilidad, robustez, comprensión, experimentación,

adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reusabilidad, eficiencia y aprendizaje) [32]. Por su parte Mynatt [8] establece que la calidad de un producto *software* se mide en función de determinados aspectos que, básicamente, son distintos para el promotor, los usuarios y los encargados de mantenimiento. El promotor o cliente del proyecto, como entidad encargada de los costes, exigirá que el producto *software* aumente la productividad de su organización, tenga un bajo coste, sea eficiente, fiable y flexible. Los usuarios exigirán que el sistema les proporcione la funcionalidad adecuada, pero también fiabilidad, eficiencia, que sea fácil de aprender, de usar y de recordar. Por último, a los encargados del mantenimiento les interesará que el sistema contenga el menor número de errores cuando se instale por primera vez, que tenga una buena documentación, que sea fiable y que esté bien diseñado.

Bertrand Meyer [42] distingue entre factores de calidad externos (tales como corrección, robustez, extensibilidad, reutilización, compatibilidad, eficiencia, portabilidad, facilidad de uso, funcionalidad y oportunidad) e internos (modularidad, legibilidad, etc.), si bien los factores externos son los que importan en última instancia, debido a que son los únicos percibidos por el usuario, son los factores internos los que aseguran el cumplimiento de los primeros. En cualquier caso, la optimización de todos estos aspectos es difícil ya que algunos son excluyentes. Además, la relación entre los costes y estos atributos no es lineal, de forma que pequeñas mejoras o variaciones en cualquiera de ellos pueden ser demasiado caras. Barry B. Boehm [43, 44] afirma que aplicar con éxito la Ingeniería del Software requiere una continua resolución de una variedad de metas importantes, pero conflictivas en la mayoría de sus casos.

En muchas ocasiones la Ingeniería del Software se ha querido limitar al desarrollo de grandes proyectos informáticos, pero si el fin último es conseguir un producto *software* de calidad, todo desarrollo *software* ha de seguir un proceso.

The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework that must be established for effective delivery of software engineering technology [5] (p. 15).

Un proceso es una colección de actividades, acciones y tareas que se llevan a cabo cuando se quiere crear un producto. Las actividades se orientan a conseguir un objetivo de grano grueso, una acción implica un conjunto de tareas que tienen como resultado un producto complejo, como por ejemplo un modelo arquitectónico, por último, una

tarea se centra en un objetivo pequeño, pero bien definido, que da lugar a un resultado tangible.

En el contexto de la Ingeniería del Software, un proceso no es una prescripción rígida de cómo se construye un sistema *software*. Debe dar una aproximación adaptable que permita al equipo de ingenieros hacer su trabajo. Como hay muchos tipos diferentes de *software*, no existe un proceso *software* universal. No obstante, cualquier proceso debe incluir, de alguna manera, las cuatro actividades principales de la Ingeniería del Software [32] (p.44):

1. *Especificación del software*. Dónde se definen la funcionalidad del *software* y sus restricciones.
2. *Desarrollo del software*. Se produce el *software* que cumple con las especificaciones.
3. *Validación del software*. Se debe asegurar que el *software* cumple con lo que el cliente espera.
4. *Evolución del software*. El *software* debe evolucionar para cumplir con las necesidades cambiantes del cliente.

Estas u otras actividades (por ejemplo, en [5] se proponen cinco – comunicación, planificación, modelado, construcción y despliegue) definirían un marco de proceso para la Ingeniería del Software, que se completa, además, con un conjunto de actividades generales que se aplican de forma transversal a todo el marco de proceso, como son, entre otras, el seguimiento y control del proyecto, el aseguramiento de la calidad del *software*, las revisiones técnicas o la gestión de la configuración del *software*.

Este marco de proceso *software* queda definido internacionalmente con el estándar ISO/IEC/IEEE 12207, cuya versión de 2008 [45] ha quedado recientemente sustituida por la versión de 2017 [46]. Este estándar ISO/IEC/IEEE 12207:2017, relativo a los procesos del ciclo de vida del *software*, se aplica a la adquisición de sistemas de *software*, productos y servicios, al suministro, desarrollo, operación, mantenimiento y eliminación de productos de *software* o componentes de *software* de cualquier sistema, ya sea que se realice interna o externamente a una organización. Se incluyen aquellos aspectos de la definición del sistema necesarios para proporcionar el contexto de los productos y servicios de *software*. También proporciona procesos que pueden emplearse para definir, controlar y mejorar los procesos del ciclo de vida del *software* dentro de una organización o de un proyecto. Los procesos, actividades y las tareas de este

estándar se pueden aplicar durante la adquisición de un sistema que contenga *software*, ya sea solo o en conjunto con el estándar ISO/IEC/IEEE 15288:2015 [47] relativo a los procesos del ciclo de vida del sistema. El mapa de procesos del ciclo de vida del *software* propuesto por el estándar ISO/IEC/IEEE 12207:2017 se puede ver en la Figura 7.2.

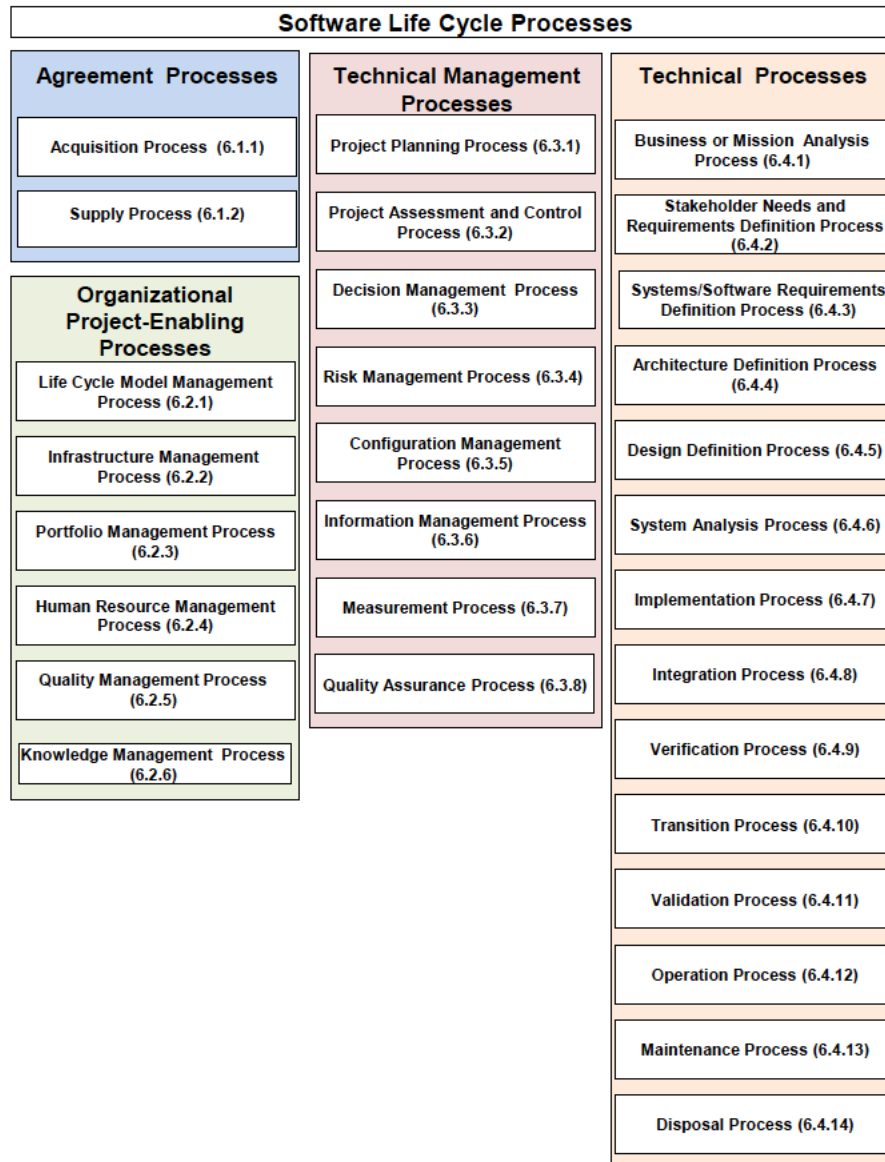


Figura 7.2. Procesos del ciclo de vida del *software* según el estándar ISO/IEC/IEEE 12207:2017. Fuente: [46] (p. 21)

La Ingeniería del Software está compuesta de pasos que abarcan los métodos, las herramientas y los procedimientos. Esos pasos constituyen lo que se ha dado en llamar ciclo de vida de un proyecto *software*. Un ciclo de vida representa la evolución de un sistema, producto, proyecto o de cualquier otra entidad construida por los seres humanos desde su concepción hasta su retirada [48]. Formalmente, el ciclo de vida del *software* se puede definir como las distintas fases por las que pasa el *software* desde que

nace una necesidad de mecanizar un proceso hasta que deja de utilizarse el *software* que sirvió para ese objetivo, pasando por las fases de desarrollo y explotación [26].

No existe una única forma de organizar las fases del ciclo de vida del *software*, la forma más natural de hacerlo sería ordenar sus fases de manera secuencial según ocurren en el tiempo, pero esto se ha demostrado que no es realista por la naturaleza evolutiva del *software*, por ello se le da un orden lógico, es decir, se representan de una manera comprensible, pero dicho orden no implica que ocurra exactamente de esa forma en el tiempo. A esta manera de representar las fases de un ciclo de vida o, lo que es lo mismo, las fases de un proceso, se denomina modelo de ciclo vida o modelo de proceso. Por tanto, no existe un único modelo de ciclo de vida o de proceso. A lo largo de los años han ido apareciendo modelos que incluyen diferentes visiones del proceso de desarrollo. Los modelos de proceso tienen su origen en intentar traer orden al caos del desarrollo de *software* [5]. Históricamente se tiene que estos modelos han aportado una cierta estructura al trabajo de los ingenieros de *software* debido a la definición de una guía efectiva para los equipos de trabajo. Sin embargo, el trabajo de los ingenieros de *software* y los artefactos que se producen están al borde del caos [49]. El borde del caos se puede definir como “el estado natural entre el orden y el caos, un gran compromiso entre estructura y sorpresa” [50]. El borde del caos se puede visualizar como un estado inestable y parcialmente estructurado. Es inestable porque se siente constantemente atraído por el caos o el orden absoluto. Se tiene la tendencia a pensar que el orden es el estado ideal de la naturaleza, pero esto no siempre es cierto, el trabajo fuera del equilibrio genera creatividad, procesos auto-organizados y rendimientos crecientes [51]. El orden absoluto significa la ausencia de variabilidad, que podría ser una ventaja en entornos impredecibles. El cambio ocurre cuando hay alguna estructura para que el cambio pueda organizarse, pero sin tanta rígido como para que no pueda ocurrir. Demasiado caos, por otro lado, puede hacer que la coordinación y la coherencia sean imposibles. La falta de estructura no siempre significa desorden. Las implicaciones filosóficas de este argumento son significativas para la Ingeniería de Software. Cada modelo de proceso trata de encontrar un equilibrio entre la necesidad de impartir orden en un mundo caótico y la necesidad de adaptarse cuando las cosas cambian constantemente.

En la [Figura 7.3](#) se puede ver una clasificación bidimensional de la tecnología basada en la clasificación de Charles Perrow [52]. En el eje de las ordenadas se presenta la

capacidad de análisis del problema, con los valores discretos de bien o mal definido, mientras que en el eje de abscisas se recoge la variabilidad de la tarea, que representa el número de excepciones esperadas. La Ingeniería del Software, a diferencia de otras formas de ingeniería, presenta problemas de toma de decisión en más de un cuadrante. La definición borrosa del problema y un número alto de excepciones caracterizan las fases de análisis de requisitos y de diseño, que deben servir de base para que la generación de código y prueba se puedan incluir en el cuadrante de la ingeniería. El mayor número de actividades van a recaer en el cuadrante de actividades no rutinarias, lo que va a requerir de personal altamente cualificado, una baja formalización y centralización, una alta demanda de procesamiento de información y una importante necesidad de coordinación del equipo de trabajo.

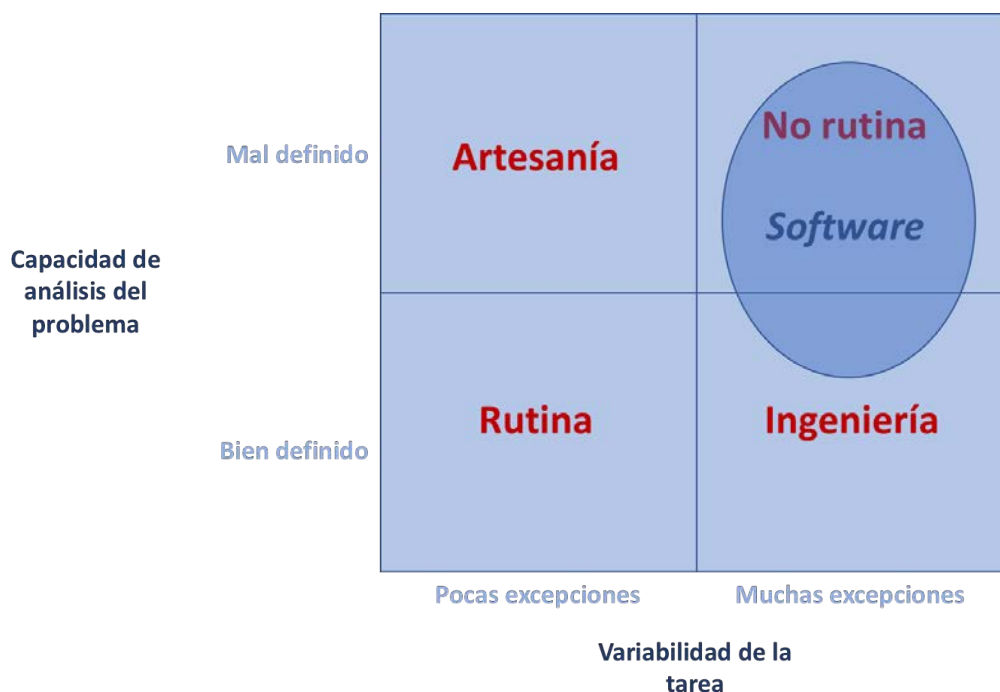


Figura 7.3. Clasificación de la tecnología. Fuente: Basado en [49]

Los modelos de proceso primigenios como el ciclo de vida en cascada [53, 54], el modelo en cascada con prototipado [55, 56], el modelo de programación automática [57], el modelo incremental [58] o el modelo en espiral [59, 60] entre otros, dieron paso a los modelos iterativos e incrementales [32]. Esa evolución coincidió con la toma en consideración del concepto de proceso del *software* [61], que pretendía englobar en un único marco los aspectos puramente de desarrollo y los de gestión. El ciclo de vida pasa a ser entonces un componente básico del proceso del *software*, pero no el único. Entre todos los modelos de proceso, el llamado Proceso Unificado [62], con una definición específica originalmente de la empresa *Rational Software* y posteriormente de *IBM*,

Rational Unified Process [63], se considera el representante más genuino de los llamados procesos pesados (*heavyweight*). Ese calificativo le fue otorgado por los defensores de una aproximación ágil [64] y que, desde su aparición en los inicios de la década pasada, ha supuesto una revolución en la forma de desarrollar *software* con la propuesta de métodos ágiles, tales como Extreme Programming [65, 66], Scrum [67, 68] o DSDM [69] entre otros. En [70] se puede consultar una línea de tiempo con las principales prácticas ágiles, lo que supone una traza temporal desde sus raíces.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

While there is value in the items on the right, we value the items on the left more.

Agile Manifesto [64].

En base al *Agile Manifesto*, se definen 12 principios [64]:

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4. *Business people and developers must work together daily throughout the project.*
5. *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
7. *Working software is the primary measure of progress.*

8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
9. *Continuous attention to technical excellence and good design enhances agility.*
10. *Simplicity – the art of maximizing the amount of work not done – is essential.*
11. *The best architectures, requirements, and designs emerge from self-organizing teams.*
12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Si un proceso *software* marca las actividades y tareas que deben realizarse en un proyecto, los métodos indican cómo desarrollar dichas tareas. Una metodología es el conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal [71]. El uso de una metodología permite el dominio del proceso *software*. Se puede decir que una metodología es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso particular correspondería a la Ingeniería del Software. A su vez, un método, es un procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla. Es un conjunto de técnicas, herramientas y tareas que, de acuerdo a un enfoque metodológico, se aplican para la resolución de un problema.

Desde el punto de vista específico de la Ingeniería del Software, la metodología describe cómo se organiza un proyecto, el orden en el que la mayoría de las actividades tienen que realizarse y los enlaces entre ellas, indicando asimismo cómo tienen que realizarse algunas tareas proporcionando las herramientas concretas e intelectuales. En concreto, se puede definir metodología de Ingeniería del Software como “*un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas*” [72].

Con una metodología se intentan cubrir las siguientes necesidades: mejores aplicaciones, mejor proceso de desarrollo y establecer un proceso estándar en una organización [6].

A una metodología se le requieren una serie de características deseables [73]:

- Un proceso de ciclo de vida completo que comprenda aspectos tanto del negocio como técnicos.

- Un conjunto completo de conceptos y modelos que sean internamente consistentes.
- Una colección de reglas y guías.
- Una descripción completa de artefactos a desarrollar.
- Una notación con la que trabajar, idealmente soportada por diversas herramientas CASE y diseñada para una usabilidad óptima.
- Un conjunto de técnicas probadas.
- Un conjunto de métricas, junto con asesoramiento sobre calidad, estándares y estrategias de prueba.
- Identificación de los roles organizacionales.
- Guías para la gestión de proyectos y aseguramiento de la calidad.
- Asesoramiento para la gestión de bibliotecas y reutilización.

Desde una perspectiva taxonómica, tradicionalmente, se pueden distinguir seis escuelas principales de pensamiento en relación con las metodologías de Ingeniería del Software:

1. *Metodologías estructuradas orientadas a datos.* Si se toma como referencia el patrón *entrada/proceso/salida* de un sistema, de forma que los datos se introducen en el sistema y este responde ante ellos para transformarlos para obtener salidas, estas metodologías se centran en la parte *entrada/salida*. En estas metodologías las actividades de análisis comienzan evaluando en primer lugar los datos y sus interrelaciones para determinar la arquitectura de datos subyacente. Cuando esta arquitectura está definida, se definen las salidas a producir y los procesos y entradas necesarios para obtenerlas. Ejemplos representativos de este grupo son los métodos JSP (*Jackson Structured Programming*) y JSD (*Jackson Structured Design*) [74-76], la construcción lógica de programas LCP (*Logical Construction Program*) [77] y el DESD (Desarrollo de Sistemas Estructurados de Datos), también conocido como metodología Warnier-Orr [78, 79].
2. *Metodologías estructuradas orientadas a procesos.* Al contrario que en el caso anterior, estas metodologías se centran más la parte de proceso del patrón *entrada/proceso/salida*. Utilizan un enfoque de descomposición descendente para evaluar los procesos del espacio del problema y los flujos de datos con los que están conectados. Este tipo de metodologías se desarrolló a lo largo de los años 70. Los creadores de este tipo de métodos fueron Edward Yourdon y Larry

- Constantine [80-82]; Tom DeMarco [83]; Chris P. Gane y Trish Sarson [84, 85]. Representantes de este grupo son las metodologías de análisis y diseño estructurado como Merise [86-88], YSM (*Yourdon Systems Method*) [89], SSADM (*Structured Systems Analysis and Design Method*) [90] (que se renombra en 2000 como *Business Development System* [91]) o METRICA v.2.1 [92] y en parte METRICA v3.0 [93].
3. *Metodologías orientadas a estados y transiciones*. Estas metodologías están dirigidas a la especificación de sistemas en tiempo real y sistemas que tienen que reaccionar continuamente a estímulos internos y externos (eventos o sucesos). Las extensiones de las metodologías de análisis y diseño estructurado de Paul T. Ward y Stephen J. Mellor [94-96] y de Derek J. Hatley e Imtiaz A. Pirbhai [97] son dos buenos ejemplos de estas metodologías.
 4. *Metodologías para el diseño basado en el conocimiento*. Utiliza técnicas y conceptos de Inteligencia Artificial para especificar y generar sistemas de información. El método KADS (*Knowledge Acquisition and Development Systems*) [98-100], la metodología IDEAL [101] o NeOn Methodology Framework [102, 103] son ejemplos de esta categoría.
 5. *Metodologías orientadas a objetos*. Se fundamentan en la integración de los dos aspectos de los sistemas de información: datos y procesos. En este paradigma un sistema se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes. El objeto encapsula datos y operaciones. Este enfoque permite un modelado más natural del mundo real y facilita enormemente la reusabilidad. Algunos representantes de este grupo son las metodologías OOA/D [104, 105], OMT (*Object Modeling Technique*) [72, 106, 107], Objectory [108], FUSION [109], MOSES [110, 111] o, en parte, METRICA v3.0 [93].
 6. *Metodologías basadas en métodos formales*. Implican una revolución en los procedimientos de desarrollo ya que, a diferencia de todas las anteriores, estas técnicas se basan en teorías matemáticas que permiten una verdadera aproximación científica y rigurosa al desarrollo de sistemas de información y *software* asociado. Un ejemplo de este tipo de metodologías puede ser OO-Method [112, 113], que está basada en el lenguaje de especificación formal OASIS (*Open and Active Specification of Information Systems*) [114-116].

En resumen, como se muestra en la Figura 7.4, la Ingeniería del Software resuelve problemas aplicando el método general de ingeniería por el que se conceptualiza el

problema, se propone una solución que se materializa en un sistema *software*. El paso de la conceptualización a la solución final se ve reflejado en un conjunto de especificaciones realizadas con diferentes lenguajes de especificación, que incluyen modelos en diferentes niveles de abstracción, desde el dominio del problema hasta el dominio de la solución. Para la realización de estos modelos se empleará un paradigma que aportará la teoría o conjunto de teorías que suministran la base para resolver los problemas. Dichos modelos representarán las diferentes vistas de todo sistema, su funcionalidad, su estructura de información y su capacidad de interoperar con otros agentes, ya sean sistemas informáticos o usuarios humanos.

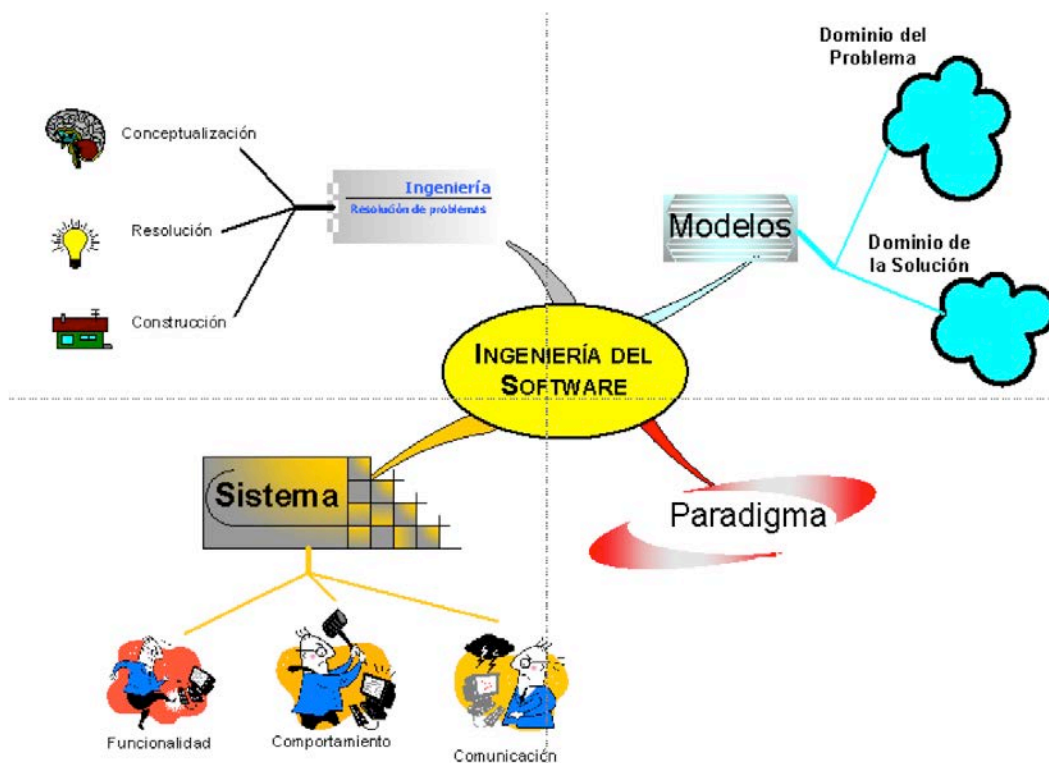


Figura 7.4. Marco conceptual de la Ingeniería del Software. Fuente: [117] (p. 11)

7.3. Evolución de la Ingeniería del Software

Pese a ser una disciplina joven en relación con otras de su misma naturaleza, la Ingeniería del Software ha evolucionado rápidamente en las últimas décadas. Este avance ha supuesto una auténtica revolución en el mundo del *software*, ya que se ha pasado de la programación artesanal al desarrollo sistemático de programas que ya pueden ser considerados productos desde el punto de vista industrial [118].

En este apartado se quiere dar un breve repaso histórico sobre cómo han avanzado las técnicas de desarrollo de programas hasta la actualidad, además de ofrecer algunas pinceladas sobre las perspectivas de futuro de la disciplina. Para ello se va a tomar como

fuente principal un artículo de Barry W. Boehm en el que hace un excelente recorrido histórico por la Ingeniería del Software [31], que queda reflejado y resumido en la Figura 7.5.

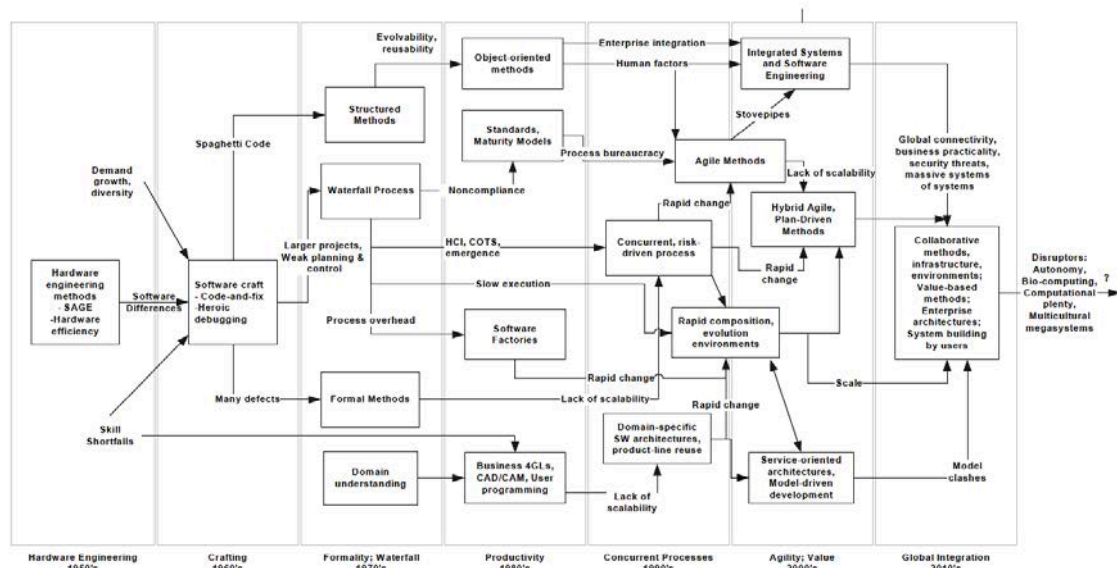


Figura 7.5. Evolución histórica de la Ingeniería del Software. Fuente: [31] (p. 16)

El recorrido comienza en la década de 1950 y de cada periodo se van a aportar unos principios atemporales, que irán precedidos de un signo '+', y una práctica que debería evitarse por quedar desfasada, la cual irá precedida de un signo '-'.

La década de 1950 parte de la tesis de que la Ingeniería del Software es equivalente a la Ingeniería del Hardware. Las principales lecciones aprendidas de esta época se pueden resumir en:

- + La Ciencia no debe descuidarse. Esta es la primera parte de la definición de ingeniería. No debería incluirse solo a las matemáticas y a la informática, sino también a las ciencias conductuales, a la economía y a la gestión. Además. Debería incluirse el uso del método científico para aprender a través de la experiencia.
- + Estudiar cuidadosamente los cambios. Los compromisos prematuros pueden acabar en desastre.
- Deben evitarse los procesos secuenciales rigurosos.

En la década de 1960 la gente descubrió que los fenómenos alrededor del *software* difieren significativamente de los fenómenos relacionados con el *hardware*. Una de las principales diferencias estriba en que el *software* es mucho más fácil de modificar que el *hardware*, por lo que las organizaciones adoptaron el modelo primitivo o modelo prueba

y error, lo que dio lugar a una aproximación artesana de la construcción del software. Así, las aplicaciones de *software* se tornaron en más dependientes de las personas que del *hardware*.

Otra importante diferencia es que el *software* no se desgasta, lo que implica que al usarse modelos de estimación basados en la confiabilidad del *hardware* los resultados son muy pobres, asimismo el mantenimiento del *software* es un subproceso que va a diferir en esencia al del *hardware*. El producto *software* es intangible, pero conlleva un alto coste económico que es difícil establecer si fue previsto o no, con lo que aparecen los retrasos en el calendario y, cuando los retrasos aparecen, de nada sirve incluir más personas en el proyecto [119]. El *software*, generalmente, tiene muchos más estados, modos y escenarios que probar, lo que hace que sus especificaciones sean más difíciles:

In order to procure a \$5 million hardware device, I would expect a 30-page specification would provide adequate detail to control the procurement. In order to procure \$5 million worth of software, a 1500-page specification is about right in order to achieve comparable control [53].

En este contexto tuvieron lugar las dos afamadas conferencias sobre Ingeniería del Software patrocinadas por la OTAN [1, 9, 10] y que favorecieron la creación de una base de conocimiento sobre el estado de la práctica de la Ingeniería del Software en la industria y en la administración pública que fue fundamental para su mejora.

Las lecciones aprendidas de esta época se pueden resumir en:

- + Pensar fuera de la caja. La ingeniería repetitiva nunca hubiera producido las grandes innovaciones. El prototipado es una práctica de bajo riesgo y alta recompensa.
- + Se deben respetar las diferencias propias del *software*. No se puede acelerar su desarrollo indefinidamente. Como es invisible, se debe encontrar buenas maneras de hacerlo visible y significativo para las diferentes partes que se ven involucradas.
- Evitar el efecto del “llanero solitario” en el desarrollo del *software*. El último minuto de toda una noche con frecuencia no resuelve los problemas y los parches se vuelven en contra rápidamente. El trabajo en equipo es fundamental y la dependencia de los individuos insostenible.

La década de 1970 se caracteriza por la formalidad y los procesos lineales. El caos del *código espagueti* propio del enfoque artesano dio origen a la programación estructurada

[120, 121], con dos ramas principales, los métodos formales [122, 123] y los métodos estructurados descendentes [124].

Como aprendizaje y síntesis de lo ocurrido en las dos décadas anteriores surge el modelo en cascada, que se presenta en la Figura 7.6. Desafortunadamente, en parte debido a la conveniencia de contratar adquisiciones de *software*, el modelo de cascada se interpretó frecuentemente como un proceso puramente secuencial, en el que el diseño no comenzaba hasta no tener un conjunto completo de requisitos, por ejemplo. Estas malas interpretaciones fueron reforzadas por los estándares de proceso del gobierno de los EEUU [125, 126] que enfatizaron una interpretación secuencial pura del modelo de cascada.

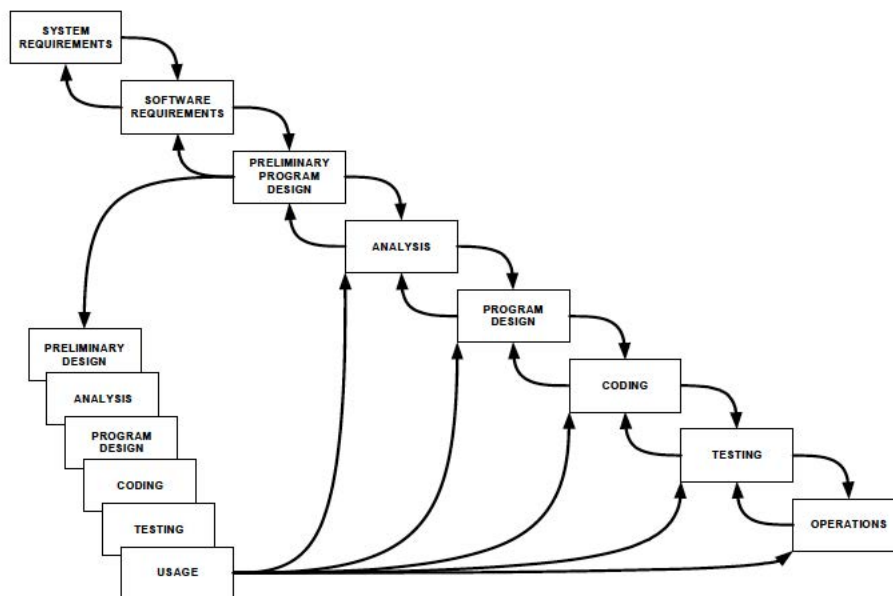


Figura 7.6. Modelo en cascada de W. Royce [53, 54]. Fuente: [31] (p. 15)

Las lecciones aprendidas de esta década se pueden resumir en:

- + Eliminar los errores en las fases tempranas del desarrollo. Mejor aún, prevenirlos antes de que ocurran.
- + Determinar el propósito del sistema. Sin una visión compartida clara es probable que se llegue al caos y a la desilusión.
- Evitar el desarrollo descendente y el reduccionismo. La orientación a objetos, el uso de componentes, la reutilización, el desarrollo ágil, etc. hacen que esta opción no sea realista para la mayoría de las aplicaciones.

La década de 1980 se focaliza en intentar solucionar los problemas de la década anterior y en mejorar la productividad y la escalabilidad de la Ingeniería del Software. Cabe

destacar que en 1984 se crea el CMU Software Engineering Institute que se encarga de desarrollar el *Software Capability Maturity Model* (SW-CMM) para evaluar la madurez de los procesos *software* de las organizaciones [24, 127]. El contenido de SW-CMM era en gran medida independiente del método, aunque mantenía fuertemente arraigada la idea de secuencialidad del modelo en cascada.

Por otra parte, se pone un gran énfasis en la integración de herramientas en entornos de soporte, tanto en los IPSE (*Integrated Programming Support Environment*) como en las herramientas CASE (*Computer-Aided Software Engineering*). De esta época viene también el concepto de *software factory*, que se emplea de forma extensiva en EEUU y Europa, pero sería en Japón donde se aplicaron con mayor efectividad [128].

Las mayores apuestas por la productividad vinieron de la mano de diferentes formas de reutilización del *software* [129], además los lenguajes orientados a objetos estimulan un rápido crecimiento de los métodos de análisis y diseño orientados objetos, los cuales acabarán convergiendo en la siguiente década gracias a la aparición en escena del *Unified Modeling Language* (UML) [130-132].

Para terminar el repaso a esta década, no puede dejar de mencionarse el artículo de Frederick P. Brooks Jr, *No silver bullet* [133], en el que muestra su escepticismo ante las innovaciones más recientes de la época y su aplicación para resolver los problemas del *software*. Para Brooks hay unas tareas esenciales en el desarrollo del *software* que inevitablemente requieren de juicio, experiencia y colaboración humana. Estas tareas esenciales tienen una relación directa con los retos del *software* para crear soluciones productivas: complejidad, adaptación, capacidad de cambio e invisibilidad.

Las lecciones aprendidas de esta década se pueden resumir en:

- + Hay muchos caminos para aumentar la productividad, incluidos dotación de personal, capacitación, herramientas, reutilización, mejora de procesos, prototipos, etc.
- + Lo que es bueno para los productos es bueno para el proceso, lo que incluye a la arquitectura, la reutilización, la capacidad de composición y la adaptabilidad.
- Se debe ser escéptico sobre las *balas de plata* y las soluciones únicas para todo.

La década de 1990 se caracteriza por la eclosión de la orientación a objetos y el desarrollo de la Web [134]. La tecnología de objetos se ve fortalecida por los avances en los

patrones de diseño [135, 136], en las arquitecturas *software* [30, 137], pero especialmente por el desarrollo de UML [130-132].

El modelo en espiral [60], conducido por la gestión del riesgo, es una propuesta de un proceso que soporta la ingeniería concurrente, que se vería completada con una mayor elaboración de las actividades de gestión del riesgo del *software* [138, 139] y el uso de la teoría *win-win* [140].

Otra de las contribuciones importantes a la ingeniería concurrente viene de la mano del *software open source*, con hitos significativos en la fundación de la Free Software Foundation, la definición de la licencia GNU *General Public License* [141], la aparición en escena del sistema operativo Linux [142, 143] o el establecimiento del consorcio World Wide Web (<https://goo.gl/DQWDK6>, ver Figura 7.7). El modelo de proceso *software* detrás de este movimiento abierto se presenta en el libro de Eric Steven Raymond *The cathedral and the bazaar* [144] y en [145] se tiene disponible un estudio empírico del ciclo de vida de proyectos abiertos a la comunidad.



Figura 7.7. Línea de tiempo del World Wide Web Consortium. Fuente: <https://goo.gl/ie1cyS>

La década de los noventa también destaca por la preocupación por el incremento de la usabilidad de los productos *software* [146, 147].

Las lecciones aprendidas de esta década se pueden resumir en:

- + El tiempo es dinero. La gente generalmente invierte en *software* para obtener un retorno positivo. Cuanto antes se envía el *software*, antes se obtiene el retorno, siempre que se tenga una calidad satisfactoria.
- + Hacer que el *software* sea útil para las personas. Esta es otra parte de la definición de ingeniería.

- Sé rápido, pero no te apresures. Los hitos iniciales ambiciosos generalmente dan como resultado especificaciones incompletas e incompatibles, además de muchas repeticiones.

La primera década del siglo XXI se caracteriza por la agilidad y el valor. Se continua con una tendencia hacia el desarrollo rápido de aplicaciones y una aceleración del ritmo de cambio en las tecnologías de la información. El final de la década de los noventa vio emerger diferentes métodos ágiles, los cuales enfatizan la mejora de la usabilidad y que el incremento de funcionalidad responda a las preferencias y el valor añadido de sus usuarios.

La tendencia al cambio frecuente en los requisitos es incompatible con las prácticas propias de los procesos clásicos, representados por el modelo secuencial en cascada y los cálculos de programación formal; y con modelos de madurez de procesos que enfatizan la repetición y la optimización [148, 149]. En su lugar, se necesitan modelos de proceso más adaptativos [150] y conducidos por la gestión del riesgo [60], que resulten también atractivos para la mejora de los procesos *software* en las organizaciones pequeñas [151]. Es interesante mencionar aquí la teoría de la *Value-Based Software Engineering* (VBSE) [152], que proporciona un marco para determinar qué partes dinámicas de bajo riesgo de un proyecto se abordan mejor mediante métodos más ágiles y qué partes de mayor riesgo y más estabilizadas se abordan mejor con métodos más planificados. Esta síntesis es cada vez más importante a medida que el *software* se vuelve más crítico para el producto o para la misión, y así que las organizaciones pueden optimizar el tiempo de lanzamiento al mercado.

Otra de las tendencias que cobró fuerzas con el nuevo siglo, aunque a día de hoy sigue sin ser una realidad en el desarrollo industrial de *software*, es desarrollo conducido por modelos o *Model-Driven Development* (MDD) [153-155]. Si bien la generación de programas a partir de modelos de alto nivel mediante transformaciones formales se desarrolla principalmente en las décadas de 1980 y 1990, es en esta década cuando tiene un repunte gracias a la consolidación de los lenguajes de modelado de muy alto nivel, no necesariamente formales, generalmente orientados a objetos, cuyo representante más genuino es el UML, que ha sido el detonante de la búsqueda de mecanismos para generar código ejecutable a partir de dichos modelos. Surge así la Ingeniería del Software Dirigida por Modelos [156], patrocinada por los grandes actores de la industria del *software*, y que tiene en el Object Management Group (<https://goo.gl/DLUAFy>) uno

de sus mayores referentes con su propuesta *Model Driven Architecture* (MDA) [157, 158]. MDA propone una jerarquía de niveles de abstracción en la que, partiendo de modelos muy abstractos, se va descendiendo hasta la implementación a base de transformaciones intra o inter nivel. La automatización de dichas transformaciones lleva a procesos de desarrollo en los que la implementación se genera automáticamente.

Las lecciones aprendidas de esta década se pueden resumir en:

- + Si el cambio es rápido, la adaptabilidad supera la capacidad de repetición.
- + Considerar y satisfacer el valor de todas las proposiciones de los diferentes *stakeholders*. Si se descuidan las partes interesadas en el éxito o en la criticidad del proyecto, generalmente contraatacarán o se negarán a participar, haciendo que todos sean perdedores.
- Evitar las tecnologías que nunca cambian su estado de *promesa* o *emergente*.

Para la década actual, 2010, y el futuro a corto medio plazo, Barry W. Boehm [159] identifica 10 tendencias, las 8 primeras se pueden clasificar de continuistas y no sorprendidas, mientras que las 2 últimas se podrían definir como tendencias comodín:

1. *Integración creciente en la Ingeniería del Software y la Ingeniería de Sistemas*. Por motivos históricos, inicialmente, ambas disciplinas tuvieron una evolución independiente. Pero a medida que el ritmo de cambio y la complejidad de los sistemas aumenta, volviéndose más intensivos en cuanto a su relación con el usuario y el *software* toma un papel mucho más protagonista en cualquier sistema, aunque este tenga un algo componente de *hardware*, hace necesario que el ingeniero de *software* salga del área proteccionista al que lo habían relegado las aproximaciones secuenciales de los procesos clásicos. Existen diferentes estándares y guías que enfatizan la necesidad de integrar los procesos de la ingeniería de sistemas y de *software*, con los aspectos de ingeniería de *hardware* y aquellos otros relacionados con las personas, como el *Integrated Capability Maturity Model* (CMMI) [160], el estándar ISO/IEC/IEEE 12207:2017(E) [46] o el estándar ISO/IEC/IEEE 15288:2015(E) [47]. Todos ellos enfatizan las prácticas de la ingeniería concurrente de requisitos y soluciones, integrando el desarrollo del producto y del proceso, con procesos basados en la gestión de riesgos en lugar de en el desarrollo de documentación, de forma que se definan nuevos hitos de los procesos que permitan la sincronización efectiva y la estabilización de los procesos concurrentes [161, 162].

2. *Un mayor énfasis en los usuarios y en el valor.* La realidad del *software* actual es que muchos de sus requisitos son más emergentes que predefinidos. Los usuarios siempre sufrirán del síndrome IKIWISI (*I'll Know It When I See It*), pero realmente en estos momentos sus prioridades cambian con el tiempo, siguiendo, en muchos casos, la jerarquía de necesidades de Maslow [163], en la que las necesidades insatisfechas de los niveles inferiores son las más prioritarias, pero pierden su prioridad una vez que se han satisfecho. Harold R. Booher [164] destaca el aumento del interés en la integración de los aspectos humanos en la Ingeniería de Sistemas. Los requisitos emergentes son incompatibles con los procesos clásicos. Se hacen necesarios los procesos flexibles y adaptativos, que refleja la teoría VBSE [165] o la ruta de los procesos *software* [166].
3. *Incremento de la criticidad los sistemas *software* intensivos y de la necesidad de su confiabilidad.* Lo que va a involucrar las cuatro siguientes tendencias.
4. *Cambio cada vez más rápido.* Los métodos ágiles tienen un campo de aplicación muy claro en los sistemas *software* intensivos y en su constante evolución y cambio. Las técnicas que permiten que los métodos ágiles funcionen bien en proyectos de tamaño pequeño o mediano [167] pueden tener serias dificultades cuando se aplican en proyectos más grandes y más críticos [168]. La Figura 7.8 identifica 5 dimensiones clave en las que los métodos ágiles y los métodos basados en planificaciones se comportan mejor: tamaño del proyecto, criticidad de la misión, competencia del personal, dinamismo del proyecto y cultura organizacional. Los proyectos que se encuentran en los dos niveles más próximos al centro de cada dimensión están dentro del dominio de los métodos ágiles, no siendo tan adecuados los métodos tradicionales. La solución radica en buscar propuestas mixtas para abordar los diferentes tipos de proyectos.

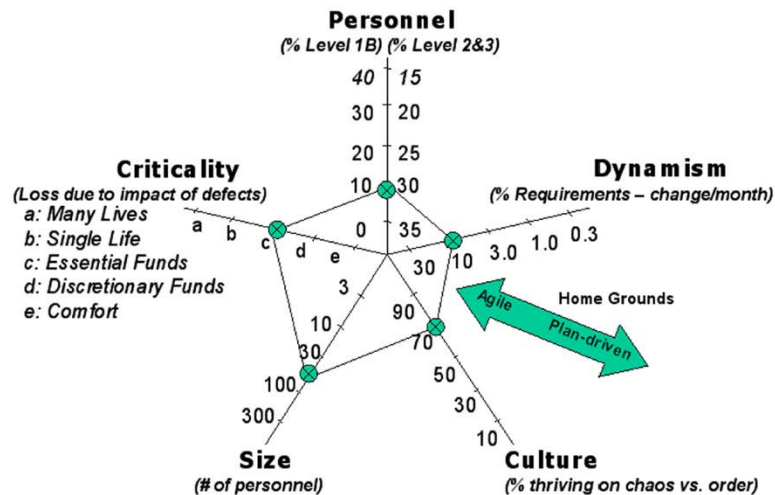


Figura 7.8. Dimensiones clave para determinar cómo afrontar un proyecto. Fuente: [159] (p. 6)

5. *Incremento de la globalización y de las necesidades de interoperabilidad.* El desarrollo de proyectos de forma colaborativa en un ámbito global es uno de los grandes retos para el futuro inmediato [169, 170]. La interoperabilidad [171] se convierte en otra de las propiedades clave para cambiar el modelo de los sistemas de información y evolucionar hacia los ecosistemas *software* [172-174].
6. *Aumento de la complejidad.* Tradicionalmente (e incluso recientemente para algunas formas de métodos ágiles), los sistemas y los procesos de desarrollo de *software* eran recetas para sistemas independientes con altos riesgos de interoperabilidad inadecuada. La experiencia ha demostrado que tales colecciones de sistemas causan retrasos inaceptables en el servicio, planes descoordinados y conflictivos, decisiones ineficaces o peligrosas e incapacidad para hacer frente a un cambio rápido. La lista priorizada de los 10 riesgos más importantes para abordar en los desarrollos complejos actuales son: (1) gestión de adquisiciones y dotación de personal; (2) viabilidad de los requisitos y de la arquitectura; (3) calendarios alcanzables; (4) integración del proveedor; (5) adaptación al cambio rápido; (6) la capacidad de alcanzar los sistemas y la calidad del *software*; (7) integración de productos y actualización electrónica; (8) Componentes COTS (*Commercial-Off-The-Shelf*) y viabilidad de reutilización; (9) interoperabilidad externa; y (10) preparación tecnológica.
7. *COTS, reutilización y retos de la integración de aplicaciones legadas.* Muchos sistemas de información se basan en composición de servicios, cambiando hacia el concepto de ecosistema tecnológico, en el que interoperan componentes y se

desarrollan aquellas aplicaciones necesarias, así como los servicios oportunos para facilitar la integración del *software* legado.

8. *Computación plena*. El incremento de la capacidad de cómputo, los dispositivos portátiles y la conectividad casi universal lleva a plantearse nuevos tipos de aplicaciones en las que los usuarios no solo son las personas, lo que está permitiendo hablar de Internet de las Cosas (*Internet of Things* – IoT) [175, 176] o la Industria 4.0 [177, 178], por poner un par de ejemplos.
9. *Autonomía*. Cubre los avances tecnológicos que usan la computación plena para posibilidad que ordenadores y software para evaluar de forma autónoma situaciones y determinar las mejores formas de acción, incluyendo agentes colaborativos [179], técnicas de *machine-learning* [180] o extensiones de robots en escala de nanotecnología [181].
10. *Combinaciones de biología y computación*. Fenómenos biológicos o moleculares para resolver problemas de cómputo fuera del alcance de la tecnología basada en silicio o computación para la mejora de las capacidades físicas o mentales humanas, tal vez integradas o unidas al cuerpo humano o sirviendo como anfitriones robóticos alternativos para cuerpos humanos.

Las lecciones aprendidas de esta década se pueden resumir en:

- + Mantener el alcance controlado. Algunos sistemas pueden ser demasiado grandes y complejos.
- + Tener una estrategia de salida Administrar las expectativas, de modo que, si las cosas van mal, exista una alternativa aceptable.
- No creerse todo lo que se lee.

En resumen, algunos de los retos de investigación de la Ingeniería del Software son el cambio, las propiedades no funcionales, el *software* como servicio, los ciclos de vida no clásicos, las arquitecturas *software*, la capacidad de configuración y las especificidades de dominio[182].

7.4. Cuerpo de conocimiento de la Ingeniería del Software

Una vez que se ha introducido cual es el marco conceptual y la evolución en el tiempo de la materia, se va a perfilar de una manera más concreta el conjunto de conocimientos que entran dentro del área de influencia de la Ingeniería del Software: *su cuerpo de conocimiento*.

Una de las tareas básicas para la definición de una profesión es el establecimiento del conjunto de conocimientos que el profesional debe poseer para el adecuado ejercicio de su labor profesional. Este cuerpo de conocimiento es fundamental para constituir el resto de los elementos que conformarán la profesión, esto es, una propuesta curricular y una política de certificación de los estudios y de los profesionales.

Ante esta necesidad se han propuesto diferentes alternativas, como pueden ser el SWEBOK (*Software Engineering Body of Knowledge*) de IEEE Computer Society [183-185], el SE-BOK propuesto por el WGSEET (*Working Group on Software Engineering Education and Training*) [186, 187] o el SWE-BOK propuesto para la FAA (*Federal Aviation Administration*) [188]. Afortunadamente solo la propuesta de IEEE Computer Society ha canalizado los resultados y se ha convertido en la referencia al cuerpo de conocimiento de la Ingeniería del Software.

A continuación, se va a describir la versión 3 del cuerpo de conocimiento SWEBOK coordinado por IEEE Computer Society.

7.4.1. Orígenes y evolución del SWEBOK

De las diferentes propuestas para la creación de un cuerpo de conocimiento de la Ingeniería del Software, el proyecto SWEBOK, coordinado por IEEE Computer Society y respaldado por otras importantes compañías y organizaciones científicas como ACM, Mitre, Boeing o Rational Software Corporation entre otras, es el que en 2001 acabó siendo acatado como estándar internacional, más concretamente la versión de 2004 [184]. En 2014 se ha publicado la versión 3 del SWEBOK [185].

Los objetivos iniciales del SWEBOK fueron [189]:

1. Promover una vista consistente de la Ingeniería del Software para todo el mundo.
2. Clarificar el lugar y establecer los límites de la Ingeniería del Software con respecto a otras disciplinas, tales como la Ciencia de la Computación, la Gestión de Proyectos, la Ingeniería de Computadores o las Matemáticas.
3. Caracterizar los contenidos de la Ingeniería del Software como disciplina.
4. Ofrecer un acceso al cuerpo de conocimiento de la Ingeniería del Software.
5. Ofrecer las bases para el desarrollo de una propuesta curricular y una política de certificación, relacionadas ambas con la Ingeniería del Software.

7.4.2. Estructura del cuerpo de conocimiento propuesto en el SWEBOK

El cuerpo de conocimiento de la Ingeniería del Software propuesto en SWEBOK se ha organizado en su versión 3 en 15 áreas de conocimiento, cada una de las cuales se desarrolla en un capítulo independiente del SWEBOK. Estas áreas de conocimiento se recogen en la Tabla 7.1. Cada área de conocimiento se desglosa en una serie de tópicos con etiquetas reconocibles que se presentan en 2 (a veces 3) niveles de anidamiento. En la Tabla 7.1 se asocia a cada área de conocimiento una figura con su desglose, para así ofrecer una visión global de todo el SWEBOK versión 3.

También se identifican 7 disciplinas que intersectan con la Ingeniería del Software y que se recogen en la Tabla 7.2.

Tabla 7.1. Áreas de conocimiento del SWEBOK versión 3. Fuente: Basada en [185] (p. xxxii)

Área de conocimiento	Desglose
Requisitos del <i>software</i> (<i>Software requirements</i>)	Figura 7.9
Diseño del <i>software</i> (<i>Software design</i>)	Figura 7.10
Construcción del <i>software</i> (<i>Software construction</i>)	Figura 7.11
Prueba del <i>software</i> (<i>Software testing</i>)	Figura 7.12
Mantenimiento del <i>software</i> (<i>Software maintenance</i>)	Figura 7.13
Gestión de la configuración del <i>software</i> (<i>Software configuration management</i>)	Figura 7.14
Gestión de la Ingeniería del Software (<i>Software Engineering management</i>)	Figura 7.15
Proceso de Ingeniería del Software (<i>Software Engineering process</i>)	Figura 7.16
Modelos y métodos de la Ingeniería del Software (<i>Software Engineering models and methods</i>)	Figura 7.17
Calidad del <i>software</i> (<i>Software quality</i>)	Figura 7.18
Práctica profesional de la Ingeniería del Software (<i>Software Engineering professional practice</i>)	Figura 7.19
Economía de la Ingeniería del Software (<i>Software Engineering Economics</i>)	Figura 7.20
Fundamentos de computación (<i>Computing foundations</i>)	Figura 7.21
Fundamentos de matemáticas (<i>Mathematical foundations</i>)	Figura 7.22
Fundamentos de ingeniería (<i>Engineering foundations</i>)	Figura 7.23

Tabla 7.2. Disciplinas relacionadas con la Ingeniería del Software según el SWEBOK versión 3. Fuente: [185] (p. xxxii)

Disciplina
Ingeniería de Computadores (<i>Computer Engineering</i>)
Ciencia de la Computación (<i>Computer Science</i>)
Gestión (<i>General Management</i>)
Matemáticas (<i>Mathematics</i>)
Gestión de Proyectos (<i>Project Management</i>)
Gestión de la Calidad (<i>Quality Management</i>)
Ingeniería de Sistemas (<i>Systems Engineering</i>)

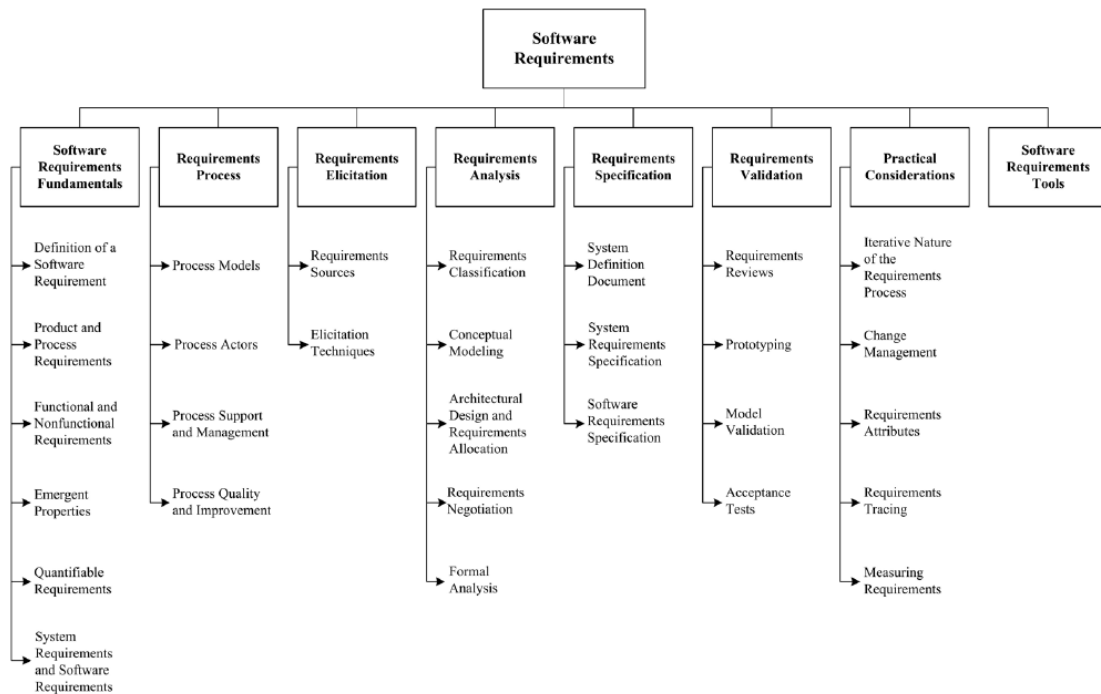


Figura 7.9. Desglose del área de conocimiento *Requisitos del software* del SWEBOK versión 3. Fuente: [185] (p. 1-2)

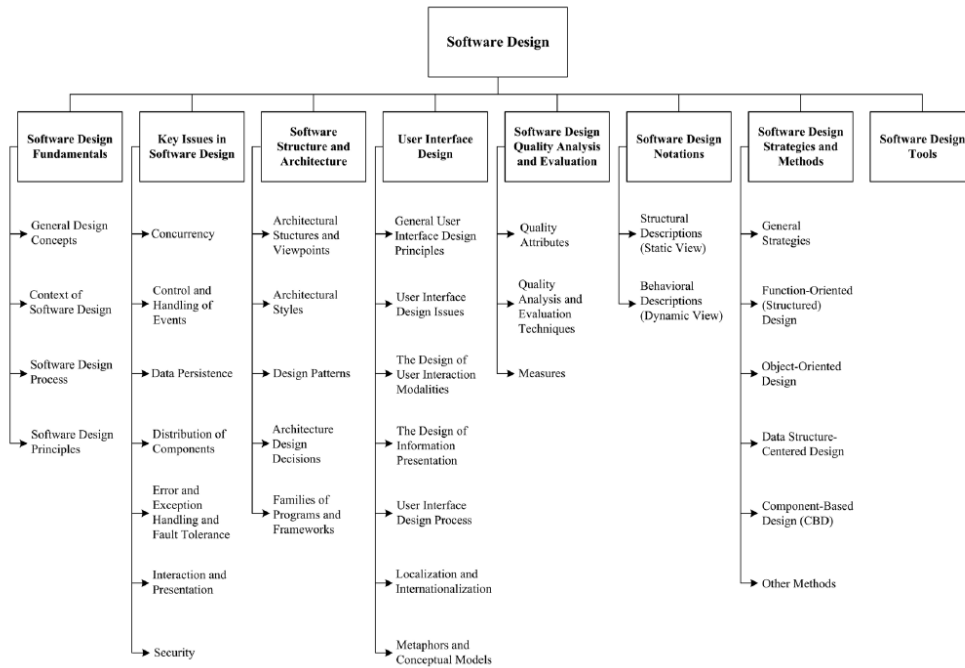


Figura 7.10. Desglose del área de conocimiento *Diseño del software* del SWEBOK versión 3. Fuente: [185] (p. 2-2)

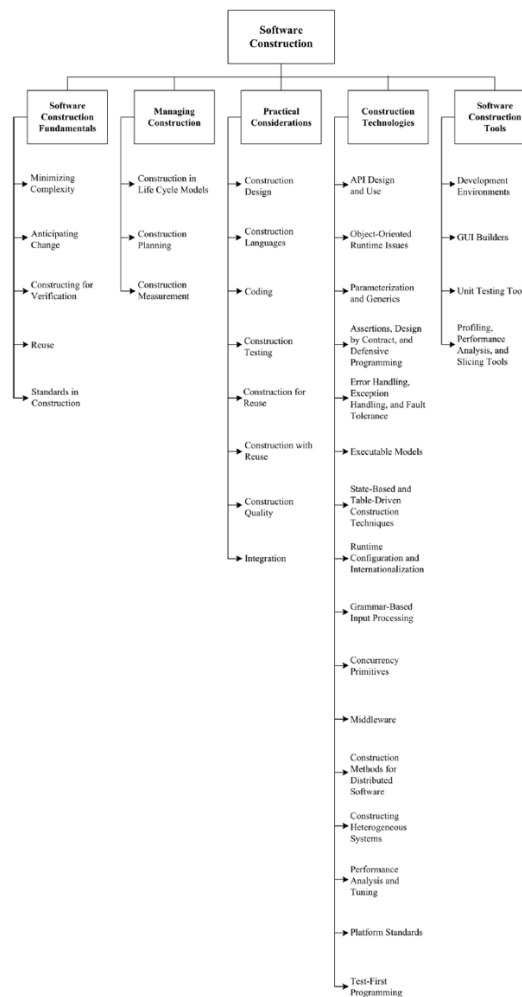


Figura 7.11. Desglose del área de conocimiento *Construcción del software* del SWEBOK versión 3. Fuente: [185] (p. 3-2)

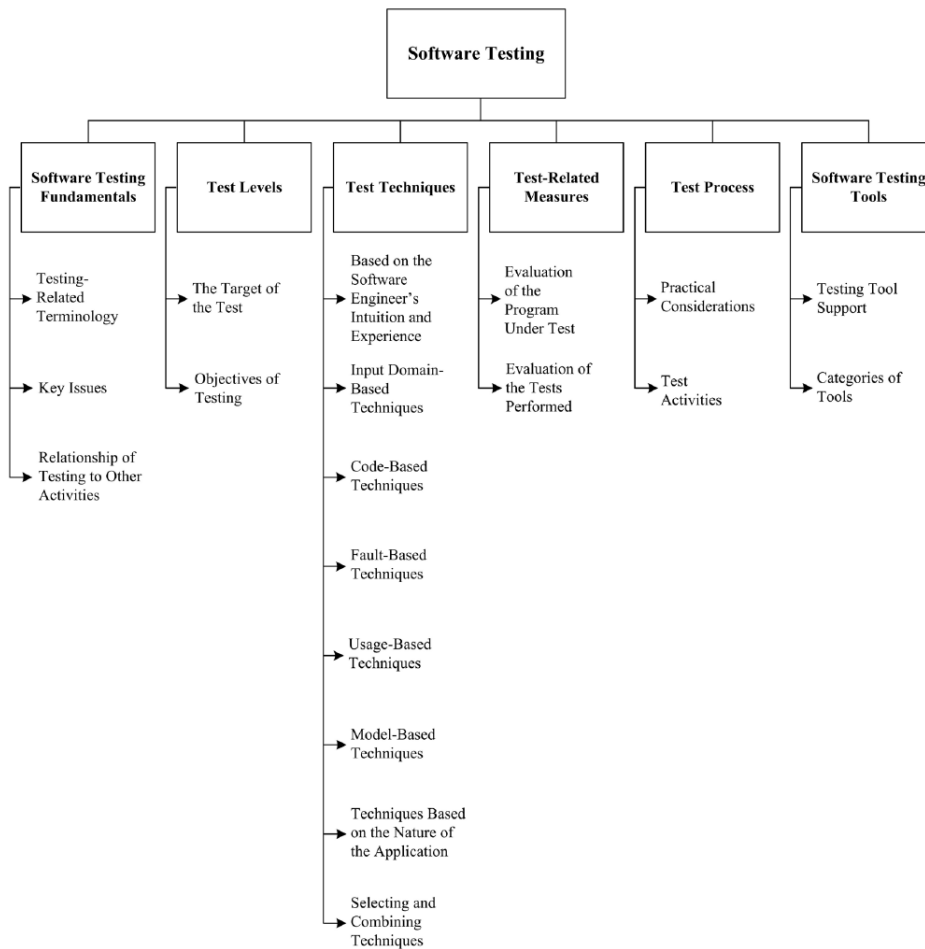


Figura 7.12. Desglose del área de conocimiento *Prueba del software* del SWEBOK versión 3. Fuente: [185] (p. 4-2)

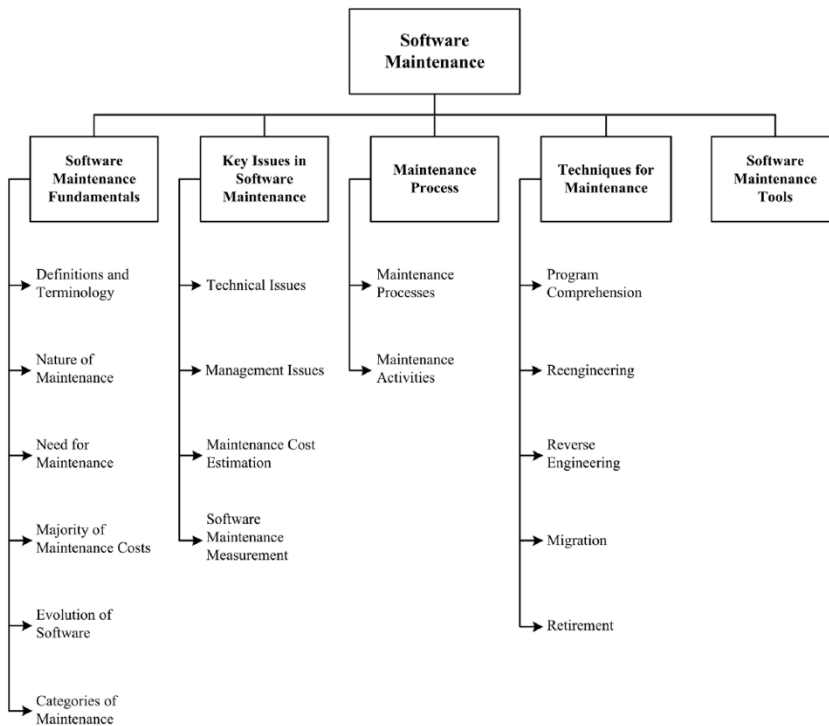


Figura 7.13. Desglose del área de conocimiento *Mantenimiento del software* del SWEBOK versión 3. Fuente: [185] (p. 5-2)

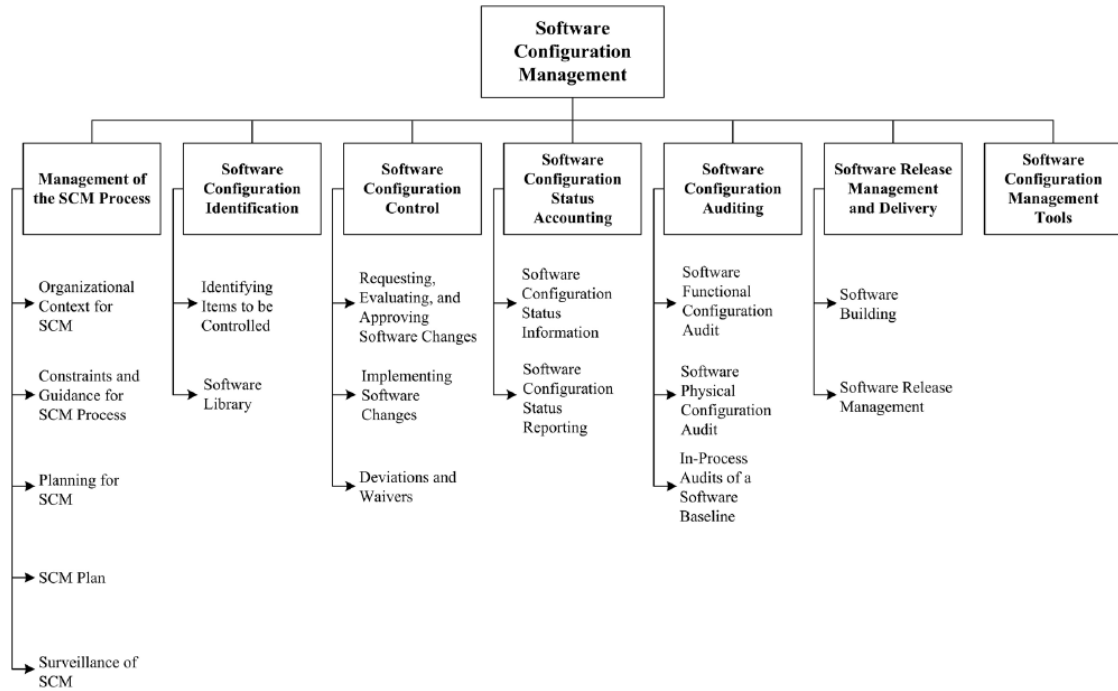


Figura 7.14. Desglose del área de conocimiento *Gestión de la configuración del software* del SWEBOK versión 3. Fuente: [185] (p. 6-2)

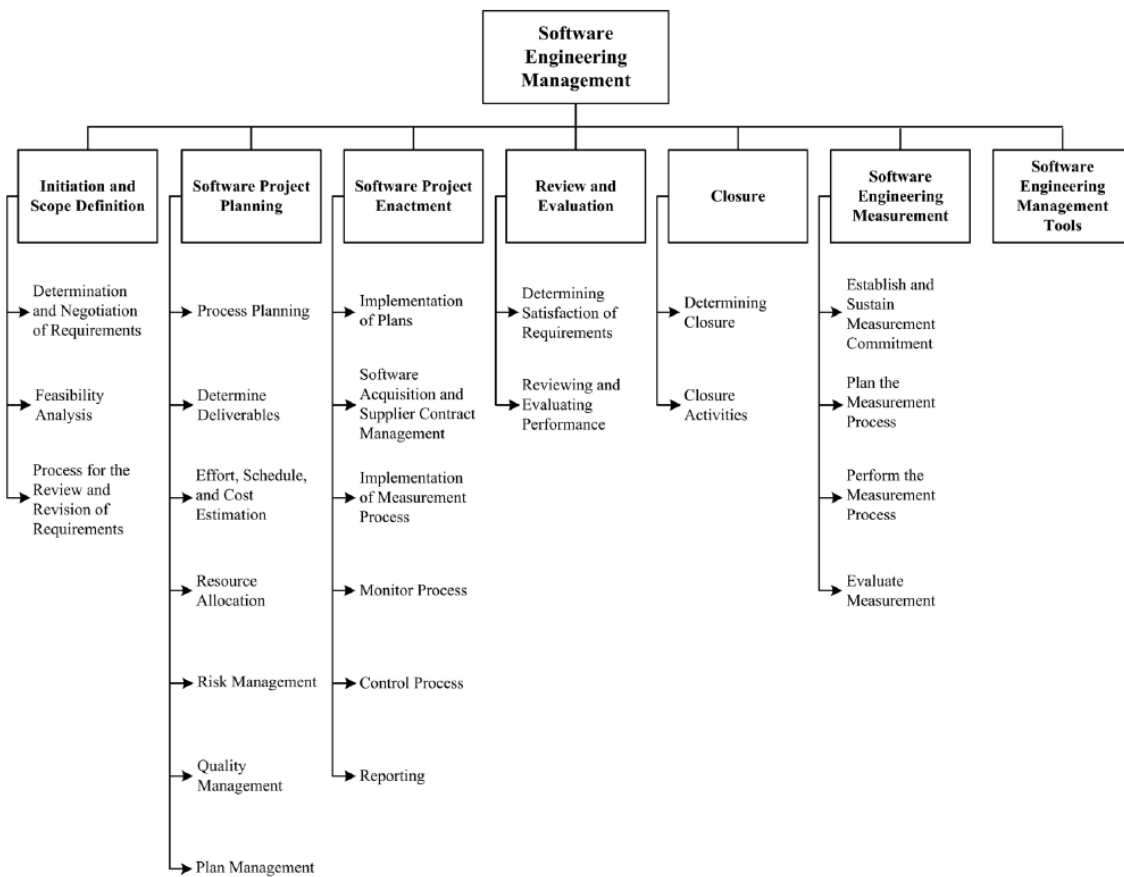


Figura 7.15. Desglose del área de conocimiento *Gestión de la Ingeniería del Software* del SWEBOK versión 3. Fuente: [185] (p. 7-2)

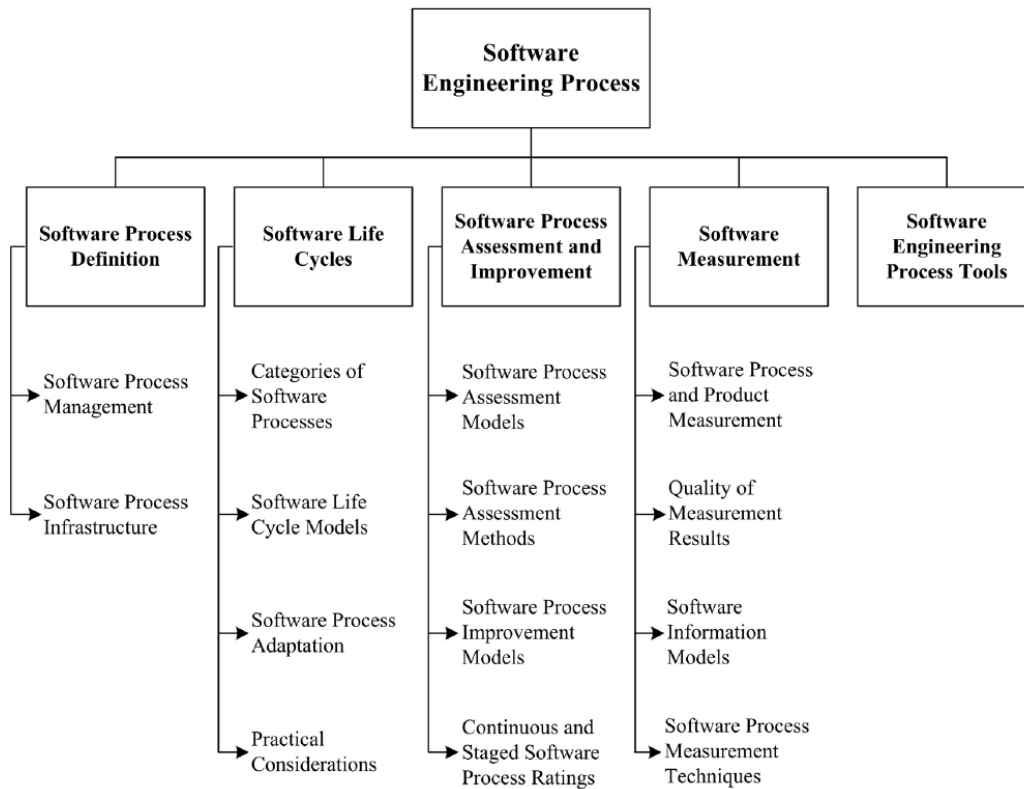


Figura 7.16. Desglose del área de conocimiento *Proceso de la Ingeniería del Software* del SWEBOK versión 3. Fuente: [185] (p. 8-2)

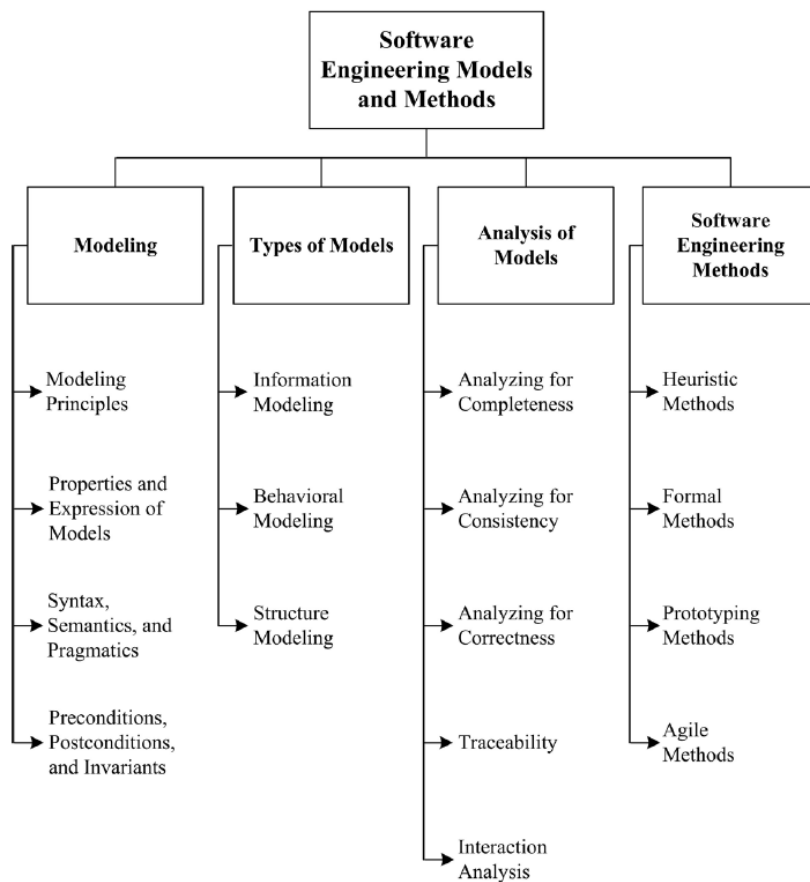


Figura 7.17. Desglose del área de conocimiento *Modelos y métodos de la Ingeniería del Software* del SWEBOK versión 3. Fuente: [185] (p. 9-2)

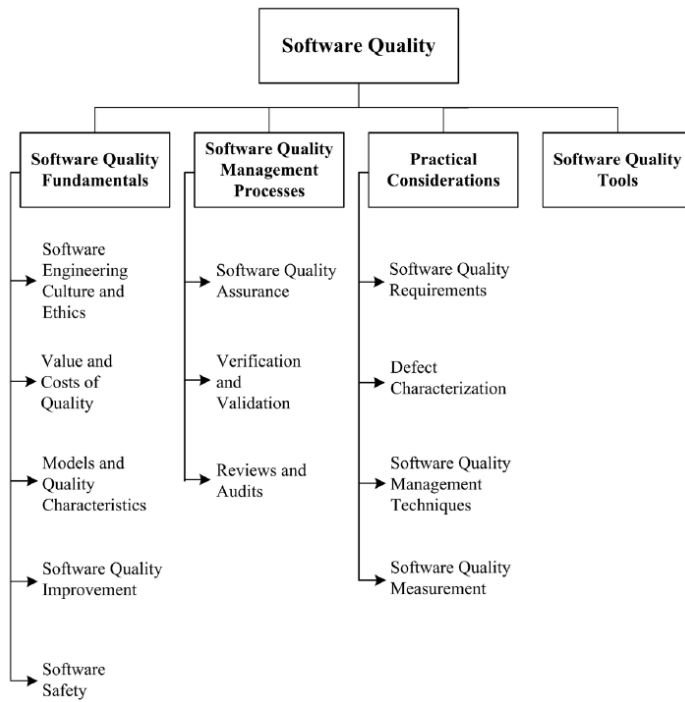


Figura 7.18. Desglose del área de conocimiento *Calidad del software* del SWEBOK versión 3. Fuente: [185] (p. 10-2)

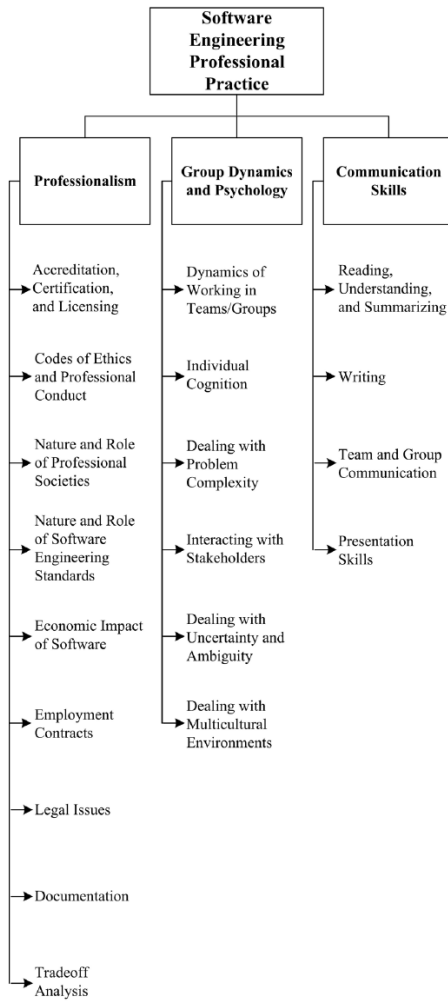


Figura 7.19. Desglose del área de conocimiento *Práctica profesional de la Ingeniería del Software* del SWEBOK versión 3. Fuente: [185] (p. 11-2)

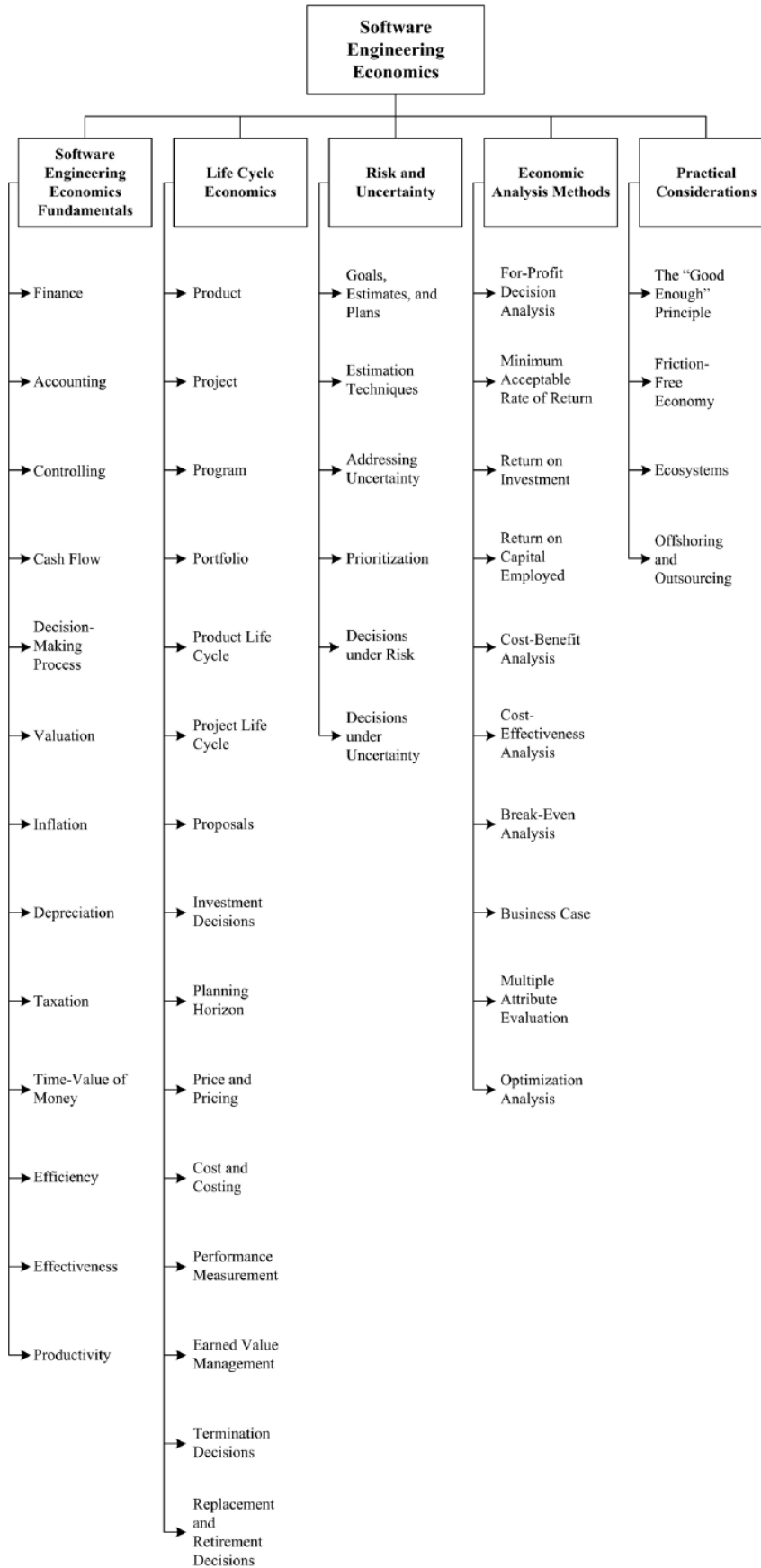


Figura 7.20. Desglose del área de conocimiento *Economía de la Ingeniería del Software* del SWEBOK versión 3. Fuente: [185] (p. 12-2)

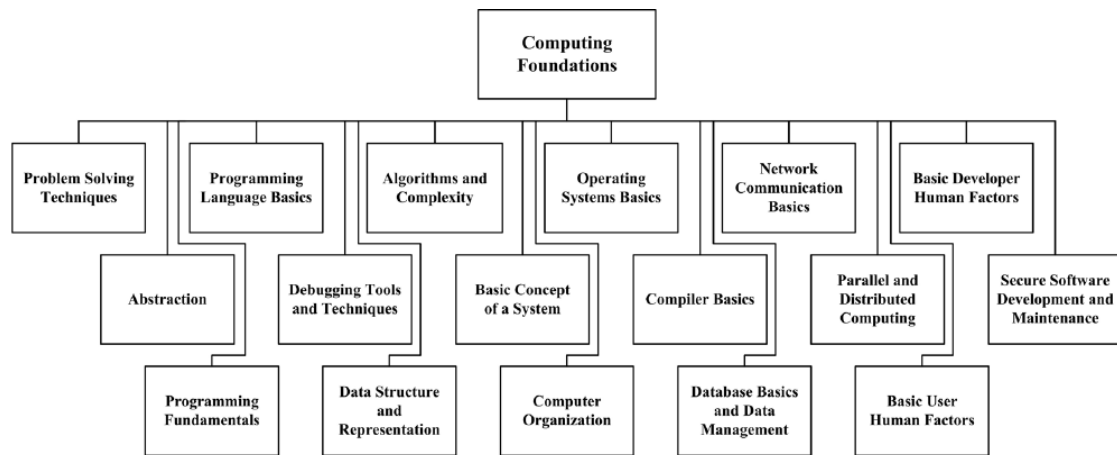


Figura 7.21. Desglose del área de conocimiento *Fundamentos de computación* del SWEBOK versión 3. Fuente: [185] (p. 13-2)

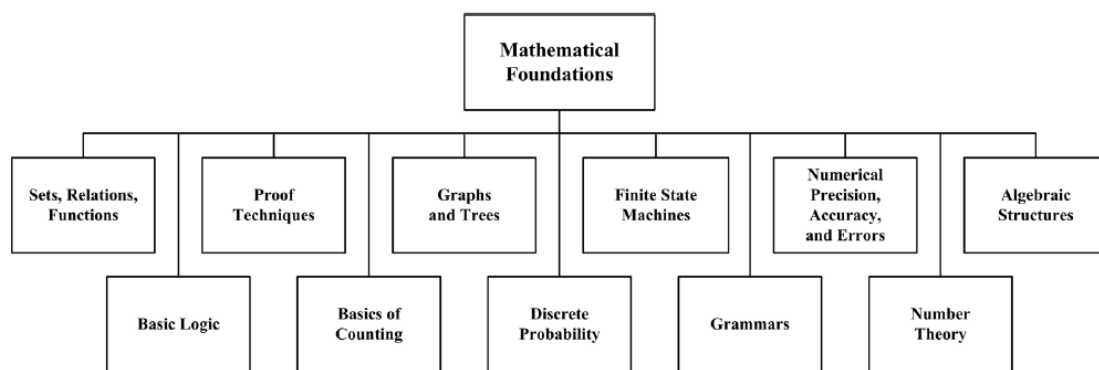


Figura 7.22. Desglose del área de conocimiento *Fundamentos de matemáticas* del SWEBOK versión 3. Fuente: [185] (p. 14-2)

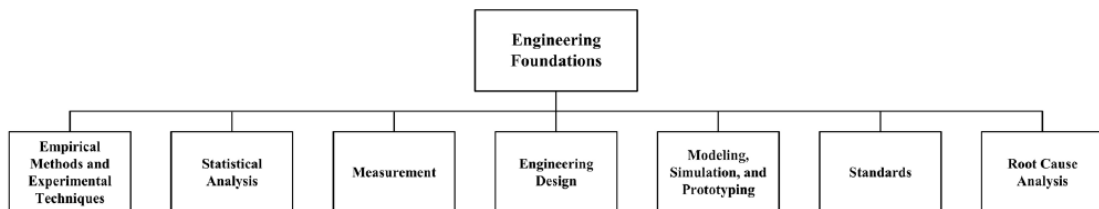


Figura 7.23. Desglose del área de conocimiento *Fundamentos de ingeniería* del SWEBOK versión 3. Fuente: [185] (p. 15-2)

Con el SWEBOK como referencia se puede decidir cómo organizar una titulación de Ingeniería de Software o, como es en el caso de España, para organizar una materia de Ingeniería de Software en una titulación de Ingeniería en Informática compuesta por varias asignaturas, cuyos contenidos pueden elegirse desde el enfoque de SWEBOK como se propone en [190].

7.5. Materia de Ingeniería del Software en los estudios de Ingeniería en Informática de la Facultad de Ciencias de la Universidad de Salamanca

En este apartado se presenta cómo se ha organizado en asignaturas la materia de Ingeniería del Software en los estudios de Ingeniería en Informática en la Facultad de Ciencias de la Universidad de Salamanca, para lo que se va a presentar los diferentes itinerarios empezando desde el Grado en Ingeniería Informática, para pasar al nivel de máster, representado por el Máster Universitario en Ingeniería Informática y el Máster Universitario en Sistemas Inteligentes. Por último, se presentarán las opciones de cerrar el periplo de educación formal con algunas opciones de Programas de Doctorado. Gráficamente, estas opciones e itinerarios se muestran en la Figura 7.24.

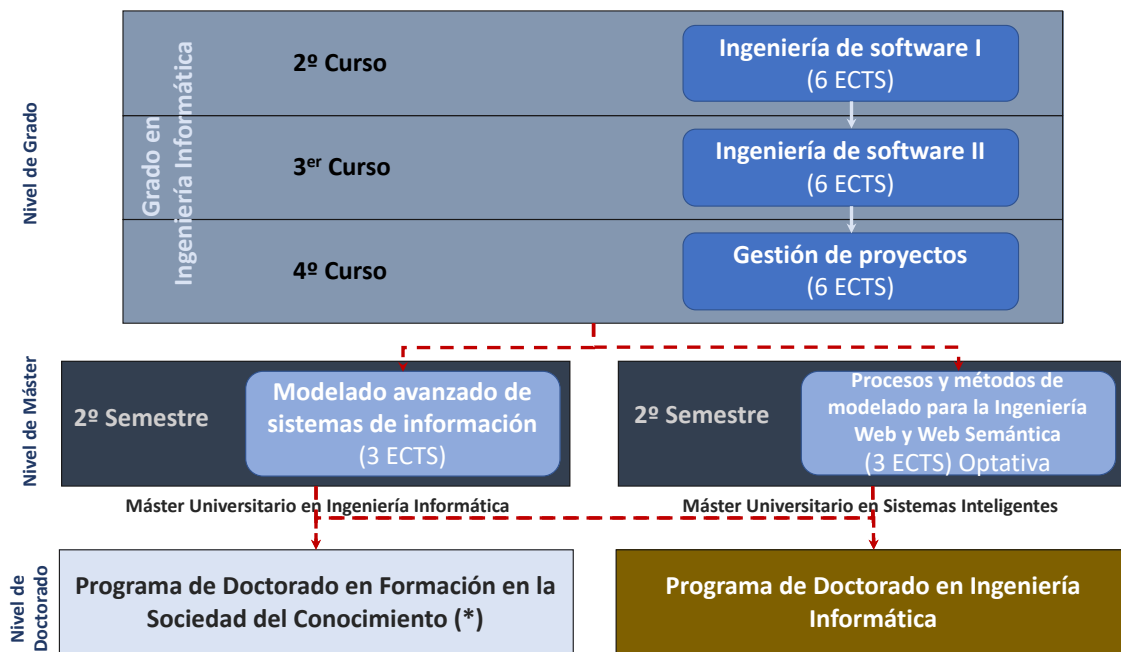


Figura 7.24. Organización de la materia Ingeniería del Software en los estudios de Ingeniería en Informática en la Facultad de Ciencias de la Universidad de Salamanca

La materia de Ingeniería del Software tiene su núcleo de definición en el Grado en Ingeniería Informática con 3 asignaturas obligatorias. La primera de ellas, *Ingeniería de Software I* [191], se imparte en el 2º semestre del 2º curso y tiene el doble objetivo de introducir los fundamentos de la materia y de introducir los modelos y métodos propios de la *Ingeniería de Requisitos*. Se continua en el 1º semestre del 3º curso con la asignatura *Ingeniería de Software II* [192], que se va a centrar en los modelos y métodos propios del *Diseño del Software*. Por último, la asignatura *Gestión de proyectos* [193], que

se imparte en el 1^{er} semestre del 4^o curso, tiene como objetivo introducir las principales técnicas para la gestión de proyectos *software*.

Una vez terminado los estudios de grado, si los estudiantes quieren continuar sus estudios con un máster universitario relacionado con la Ingeniería en Informática, la Facultad de Ciencias ofrece dos posibles opciones.

En primer lugar podría elegir seguir con el Máster Universitario en Ingeniería Informática, que es un máster orientado a un perfil profesional para formar a un ingeniero en informática según la definición oficial de la profesión descrita en [194]. En este máster hay diferentes asignaturas con tópicos incorporados en el SWEBOK, pero solo una asignatura que encajaría completamente dentro de la materia de Ingeniería del Software. Se trata de la asignatura *Modelado avanzado de sistemas de información* [195], que se imparte en el 2^o semestre del 1^{er} curso, que se centra, fundamentalmente en la Ingeniería del Software Conducida por Modelos.

La otra opción sería elegir seguir con el Máster Universitario en Sistemas Inteligentes, que tiene un perfil investigador orientado a que el estudiante continúe posteriormente con un doctorado. Este máster solo cuenta con una asignatura optativa dentro de la materia de Ingeniería del Software, *Procesos y métodos de modelado para la Ingeniería Web y Web Semántica* [196], que se imparte en el 2^o semestre del 1^{er} curso, que tiene como objetivo introducir los modelos y los métodos relacionados con la Ingeniería Web y la Web Semántica.

Una vez que se han concluido los estudios de máster universitario, se abre la opción de realizar un doctorado. Quienes hayan cursado el Máster Universitario en Sistemas Inteligentes ya conocen que la lógica continuación viene de la mano del Programa de Doctorado en Ingeniería Informática (<https://goo.gl/3m2XQe>). Para los que hayan cursado el Máster Universitario en Ingeniería Informática, al tener un perfil más profesional, el doctorado no es tan evidente, pero también tendrían la opción de continuar en dicho Programa de Doctorado. En el Programa de Doctorado en Ingeniería Informática se pueden desarrollar tesis doctorales que encajen dentro de los tópicos de la Ingeniería del Software y/o de la Ingeniería Web.

Aunque no está vinculado a la rama de Ingeniería y Arquitectura como el Programa de Doctorado en Ingeniería Informática, se ha incorporado como opción viable para los egresados de los másteres de Ingeniería en Informática el Programa de Doctorado en

Formación en la Sociedad del Conocimiento [197-200], porque, dado su carácter multidisciplinar, acepta ingenieros que quieran desarrollar su investigación aplicando de forma práctica los fundamentos de la Ingeniería del Software y de la Computación para solucionar problemas de la Sociedad del Conocimiento.

7.6. Descripción de la asignatura Ingeniería de Software I

De todas las asignaturas que conforman la materia de Ingeniería del Software descritas en el apartado 7.5, aunque se ha participado en el diseño e impartido y coordinado todas ellas, se va a elegir la asignatura Ingeniería de Software I para hacer una descripción en profundidad. El motivo es que, como ya se ha mencionado, esta asignatura representa la introducción de los fundamentos para los estudiantes del Grado en Ingeniería Informática y estos contenidos se llevan impartiendo en los diferentes planes de estudio, con independencia de su denominación y número de créditos, ininterrumpidamente desde el curso 1996-1997, primero en la Universidad de Burgos y después en la Universidad de Salamanca, desde el curso 1998-1999 hasta el actual 2017-2018.

Para realizar la descripción de esta asignatura se va a seguir un patrón basado en el propuesto en [201], utilizado en el proyecto de innovación US14/04 [202] y que se empleó también para el desarrollo de guías de asignaturas de la materia Ingeniería del Software en otros centros de la Universidad de Salamanca [203].

7.6.1. Datos básicos

En la Tabla 7.3 se recogen los datos básicos de la asignatura Ingeniería de Software I.

Tabla 7.3. Datos de la asignatura Ingeniería de Software I

Asignatura	Ingeniería de Software I
Código de Asignatura	101118
Titulación	Grado en Ingeniería Informática
Código de Titulación	2502283
Bloque formativo	Ingeniería del Software
Centro	Facultad de Ciencias
Código de Centro	37007912
Áreas de Conocimiento	Ciencia de la Computación e Inteligencia Artificial Lenguajes y Sistemas Informáticos
Departamento	Informática y Automática
Curso de inicio	2010-2011
Curso actual	2017-2018
Carácter	Obligatorio
ECTS	6
ECTS Teoría	4,5
ECTS Práctica	1,5
Unidad temporal	Segundo semestre
Coordinador de la Asignatura	Dr. D. Francisco José García Peñalvo
Profesorado Grupo A	Dr. D. Francisco José García Peñalvo Dña. Alicia García Holgado
Profesorado Grupo B	Dra. Dña. Davinia Carolina Zato Domínguez Dr. D. Jesús Fernando Rodríguez Aragón

7.6.2. Objetivos de aprendizaje

Esta es la primera asignatura que se imparte del bloque de Ingeniería del Software, por lo que en ella se le da al estudiante una visión general de la materia y se abordan las primeras actividades del proceso *software*.

Desde un punto de vista de perfil profesional, la asignatura se centra en las primeras fases del ciclo de vida de los sistemas *software*, es decir, en su concepción, planificación y análisis, lo que afecta a todos los perfiles profesionales relacionados con la gestión, consultoría y desarrollo de sistemas *software*.

Concretamente, los objetivos de aprendizaje de la asignatura son:

- O1 Conocer los elementos, la estructura y los diferentes tipos de sistemas de información.
- O2 Entender las actividades de ingeniería que componen el proceso del *software* y conocer los diferentes modelos de proceso.
- O3 Saber obtener, analizar y documentar los requisitos de un sistema *software*, para lo que se aplicarán los principios, técnicas y herramientas apropiados.
- O4 Modelar un sistema *software* en diferentes niveles de abstracción mediante el uso de un lenguaje de modelado estándar.

7.6.3. Competencias

En la Tabla 7.4 se recogen las competencias de la asignatura.

Tabla 7.4. Competencias de la asignatura Ingeniería de Software I

Tipo	Código	Competencia
Básica	CB5	Conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos, los fundamentos de su programación y su aplicación para la resolución de problemas propios de la ingeniería
Común	CC1	Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente
Común	CC2	Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social
Común	CC8	Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados
Común	CC16	Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de <i>software</i>
Tecnología Específica (Ingeniería de Software)	IS2	Capacidad para valorar las necesidades del cliente y especificar los requisitos <i>software</i> para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones

Tipo	Código	Competencia
Tecnología Específica (Ingeniería de Software)	IS4	Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones <i>software</i> sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales
Tecnología Específica (Tecnologías de la Información)	TI1	Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones
Transversal	CT1	Conocimientos generales básicos
Transversal	CT2	Conocimientos básicos de la profesión
Transversal	CT3	Capacidad de análisis y síntesis
Transversal	CT4	Capacidad de organizar y planificar
Transversal	CT5	Comunicación oral y escrita en la lengua propia
Transversal	CT8	Habilidades de gestión de la información
Transversal	CT9	Resolución de problemas
Transversal	CT10	Toma de decisiones
Transversal	CT11	Capacidad crítica y autocrítica
Transversal	CT12	Trabajo en equipo
Transversal	CT13	Capacidad de trabajar en un equipo interdisciplinar
Transversal	CT14	Responsabilidad y compromiso ético
Transversal	CT16	Capacidad de aplicar los conocimientos en la práctica
Transversal	CT17	Habilidades de investigación
Transversal	CT18	Capacidad de aprender
Transversal	CT19	Capacidad de adaptarse a nuevas situaciones
Transversal	CT20	Capacidad de generar nuevas ideas
Transversal	CT21	Habilidad para trabajar de forma autónoma y cumplir plazos
Transversal	CT22	Diseño y gestión de proyectos

7.6.4. Temario

El temario de la asignatura Ingeniería de Software I se compone de 8 temas:

- Tema 1 Introducción a la Ingeniería del Software.
- Tema 2 Sistemas de información.
- Tema 3 Modelos de procesos.
- Tema 4 Ingeniería de requisitos.
- Tema 5 Introducción al Proceso Unificado.
- Tema 6 Flujos de trabajo del Proceso Unificado.
- Tema 7 Análisis orientado a objetos.
- Tema 8 UML. Unified Modeling Language.

7.6.4.1. Tema 1 – Introducción a la Ingeniería del Software

Resumen

Se presentan los conceptos clásicos relacionados con el *software* y la Ingeniería del Software. El objetivo de este tema es tomar conciencia de la importancia de abordar la construcción del *software* desde una perspectiva de ingeniería. Se exponen los elementos constituyentes de un paradigma de desarrollo del *software*. Se ofrece una visión general del concepto de proceso y modelo de proceso *software*. Se introduce el concepto de

metodología de desarrollo como contraposición al desarrollo anárquico y artesanal de aplicaciones, tan relacionado con la tan nombrada crisis del *software*. Se termina el tema hablando de herramientas CASE.

Descriptores

Ingeniería del Software; *Software*; Aplicaciones del *software*; Crisis del *software*; Proceso *software*; Modelo de proceso; Ciclo de vida; Metodología; Método; Herramienta CASE.

Competencias

CB5; CC16; TI1; CT1; CT2; CT11; CT14; CT18.

Contenidos

1. *Software*.
2. Conceptos básicos de la Ingeniería del Software.
3. Proceso *software*.
4. Metodologías.
5. CASE.

Recursos

Recursos docentes:

- Introducción a la Ingeniería del Software [204].

Bibliografía:

1. C. Larman, UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al Proceso Unificado, 2ª ed. Madrid, España: Pearson Educación, 2003 [205] (3ª edición en inglés [206]). **Capítulo 2.**
2. M. G. Piattini Velthius, J. A. Calvo-Manzano, J. Cervera Bravo y L. Fernández Sanz, *Análisis y Diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software*. Madrid, España: Ra-ma, 2004 [6]. **Capítulos 3 y 4.**
3. S. L. Pfleeger, *Ingeniería del Software. Teoría y Práctica*. Argentina: Prentice Hall, 2002 [207]. **Capítulos 1 y 2.**
4. R. S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 7ª ed. México D. F., México: McGraw-Hill, 2010 [208] (8ª edición en inglés [5]). **Capítulos 1, 2 y 3.**
5. I. Sommerville, *Ingeniería del Software*, 9ª ed. México: Pearson Educación, 2011 [2] (10ª edición en inglés [32]). **Capítulos 1 y 2.**

Lecturas complementarias:

1. M. E. Fayad, M. Laitinen y R. P. Ward, "Thinking objectively: Software engineering in the small," *Communications of the ACM*, vol. 43, no. 3, pp. 115-118, 2000. doi: 10.1145/330534.330555. Disponible en: <https://goo.gl/KHhHaF> [209].

Artículo donde se defiende que las compañías que desarrollan proyectos *software* de pequeño tamaño también deben utilizar técnicas de Ingeniería del Software.

2. A. Fuggetta, "A Classification of CASE Technology," *Computer*, vol. 26, no. 12, pp. 25-38, 1993. doi: 10.1109/2.247645 [210].

Artículo que realiza un informe sobre la clasificación de la tecnología CASE.

3. D. Gage, "Consumer products: When software bugs bite," *Baseline. Driving Business Success With Technology*, 2003, Disponible en: <https://goo.gl/BNMvR2> [211].

Pone de manifiesto el desprestigio hacia la industria del *software*.

4. R. L. Glass, "Talk about a software crisis - not!," *Journal of Systems and Software*, vol. 55, no. 1, pp. 1-2, 2000. doi: 10.1016/s0164-1212(00)00043-1 [212].

Opinión personal del autor del artículo sobre la crisis del *software*.

5. J. L. Lions, "ARIANE 5 Flight 501 Failure," Report by the Inquiry Board, 1996. Disponible en: <https://goo.gl/nSH6Ht> [213].

Informe de las causas del fallo del lanzamiento de la lanzadera espacial Ariane 5 el 4 de junio de 1996.

6. Ministerio de las Administraciones Públicas, *Métrica v3*, Madrid, España: Ministerio de las Administraciones Públicas, 2001. [Online]. Disponible en: <https://goo.gl/FZ3aX4> [93].

Documentación en línea sobre la metodología Métrica v3.

7. L. B. S. Raccoon, "Fifty years of progress in software engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 1, pp. 88-104, 1997. doi: 10.1145/251759.251878 [214].

Presenta una visión de la evolución de la Ingeniería del Software. Presenta también el modelo de ciclo de vida Caos, comparándolo con otros modelos.

8. Risk Forum. (2018). Disponible en: <https://goo.gl/U9wifz> [215].

Foro donde se describen diversas situaciones problemáticas causadas por fallos informáticos.

9. S. Sharma y A. Rai, “CASE deployment in IS organizations,” *Communications of the ACM*, vol. 43, no. 1, pp. 80-88, 2000. doi: 10.1145/323830.323848 [216].

Informe de la presencia de la tecnología CASE en los sistemas de información de las empresas.

10. R. Singh, “The Software Life Cycle Processes standard,” *Computer*, vol. 28, no. 11, pp. 89-90, 1995. doi: 10.1109/2.471194 [217].

Visión esquemática del estándar ISO/IEC 12207 sobre los procesos que componen el ciclo de vida del *software*.

11. E. Yourdon, *Análisis Estructurado Moderno*. México: Prentice-Hall Hispanoamericana, 1993 [218].

En el capítulo 5 describe el ciclo de vida estructurado.

7.6.4.2. Tema 2 – Sistemas de información

Resumen

El objetivo de este tema es introducir el concepto de ingeniería de sistemas basados en ordenador y explicar la importancia del conocimiento de la ingeniería de sistemas para los ingenieros de *software*. Se estudiarán los componentes y estructura de los sistemas de información automatizados, así como diferentes tipos de sistemas en función de su posición en la estructura piramidal descrita previamente. Finalmente, se presentarán las diferentes actividades englobadas en la ingeniería de sistemas.

Descriptor

Sistema de información; ingeniería de sistemas; principios generales de sistemas.

Competencias

CB5; TI1; CT1; CT2; CT10; CT18.

Contenidos

1. Conceptos básicos.
2. Estructura de los sistemas de información.
3. Clasificación de los sistemas de información.
4. Principios generales de sistemas.
5. Ingeniería de sistemas.

Recursos

Recursos docentes:

- Sistemas de información [219].

Bibliografía:

1. M. G. Piattini Velthius, J. A. Calvo-Manzano, J. Cervera Bravo y L. Fernández Sanz, *Análisis y Diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software*. Madrid, España: Ra-ma, 2004 [6]. **Capítulo 1.**

Lecturas complementarias:

1. J. Chandra *et al.*, "Information systems frontiers," *Communications of the ACM*, vol. 43, no. 1, pp. 71-79, 2000. doi: 10.1145/323830.323847 [220].

Artículo que trata sobre el crecimiento y aplicación de los sistemas de información, junto a las tecnologías de la información, en dominios de aplicación que nunca habían sido considerados.

2. J. Fernández González, "Business Intelligence: Analizando datos para extraer nueva información y tomar mejores decisiones," *Novática. Revista de la Asociación de Técnicos en Informática*, vol. XXXVII, no. 211, pp. 6-7, 2011 [221].

Número especial de la revista Novática dedicado a la inteligencia de negocio. Contiene artículos de varios autores sobre diferentes aspectos de la inteligencia de negocio y su utilidad en los sistemas de información de las empresas.

3. A. J. Swartz, "Airport 95: automated baggage system?," *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 2, pp. 79-83, 1996. doi: 10.1145/227531.227544 [222].

Artículo que presenta un caso de estudio de lo que puede resultar mal en un proyecto de ingeniería de sistemas y como el *software* tiende a ser el responsable de los grandes fallos en los sistemas.

4. S. White *et al.*, "Systems engineering of computer-based systems," *Computer*, vol. 26, no. 11, pp. 54-65, 1993. doi: 10.1109/2.241426 [223].

Este artículo contiene una buena descripción de los sistemas de información basados en computadora.

7.6.4.3. Tema 3 – Modelos de proceso

Resumen

Se presentan diferentes modelos de proceso clasificados por categorías. Se parte del modelo clásico o en cascada y diferentes variantes del mismo. Posteriormente, se abordan modelos más evolucionados como pueden ser los modelos evolutivos en los que se considera la naturaleza cambiante del *software*, modelos específicos para sistemas orientados a objetos o modelos basados en reutilización centrados en el uso y desarrollo de componentes reutilizables. Asimismo, se abordan modelos más recientes tales como los procesos ágiles que enfatizan la programación frente al análisis, diseño y documentación, y modelos enfocados al desarrollo de sistemas web.

Descriptorios

Modelos de proceso; ciclo de vida; fases; modelos evolutivos; reutilización; orientación a objetos; procesos ágiles; ingeniería web.

Competencias

CC1; CC2; CC16; TI1; CT1; CT2; CT18.

Contenidos

1. Clasificación de los modelos de proceso.
2. Modelos tradicionales.
3. Modelos evolutivos.
4. Modelos para sistemas orientados a objetos.
5. Modelos basados en reutilización.
6. Procesos ágiles.
7. Modelos para la Ingeniería Web.

Recursos

Recursos docentes:

- Modelos de proceso [224].

Bibliografía:

1. M. G. Piattini Velthius, J. A. Calvo-Manzano, J. Cervera Bravo y L. Fernández Sanz, *Análisis y Diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software*. Madrid, España: Ra-ma, 2004 [6]. **Capítulo 3.**
2. S. L. Pfleeger, *Ingeniería del Software. Teoría y Práctica*. Argentina: Prentice Hall, 2002 [207]. **Capítulo 2.**
3. R. S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 7ª ed. México D. F., México: McGraw-Hill, 2010 [208] (8ª edición en inglés [5]). **Capítulos 2 y 3.**
4. I. Sommerville, *Ingeniería del Software*, 9ª ed. México: Pearson Educación, 2011 [2] (10ª edición en inglés [32]). **Capítulos 2 y 3.**

Lecturas complementarias:

1. B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah y R. Madachy, "Using the WinWin spiral model: a case study," *Computer*, vol. 31, no. 7, pp. 33-44, 1998. doi: 10.1109/2.689675 [225].

En este artículo se presenta la aplicación práctica del modelo de ciclo de vida en espiral WinWin, una extensión del ciclo de vida definido por Boehm, al que se le ha añadido las actividades de la *Teoría W* al comienzo de cada ciclo.

2. I. Gutiérrez y N. Medinilla, "Contra el arraigo de la cascada," en *Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99 (24-26 de noviembre de 1999, Cáceres - España)*, P. Botella, J. Hernández y F. Saltor, Eds. pp. 393-404, 1999 [226].

Trabajo crítico con el modelo de ciclo de vida en cascada, realizado desde la perspectiva de la complejidad de la incertidumbre en los proyectos *software*.

3. B. Henderson-Sellers y J. M. Edwards, "The object-oriented systems life cycle," *Communications of the ACM*, vol. 33, no. 9, pp. 142-159, 1990. doi: 10.1145/83880.84529 [227].

En este artículo se describe el modelo de ciclo de vida fuente para desarrollos orientados a objetos.

7.6.4.4. Tema 4 – Ingeniería de requisitos

Resumen

Es el punto de partida para un proyecto *software* y la parte más importante del proceso de desarrollo. Si los desarrolladores no conocen de forma precisa el problema a resolver, no es probable que se obtenga una solución correcta y útil. Así pues, la correcta obtención de los requisitos es uno de los aspectos más críticos de un proyecto *software*, independientemente del tipo de proyecto que se trate, dado que una mala captura de los mismos es la causa de la mayor parte de los problemas que surgen a lo largo del ciclo de vida. La ingeniería de requisitos es la parte de la Ingeniería del Software que aborda el problema de la definición de los servicios que el sistema ha de proporcionar y de establecer las restricciones operativas del mismo. Los casos de uso se han convertido en una de las técnicas de modelado más utilizadas para la determinación y documentación de los requisitos funcionales de un sistema *software*. En este tema se presentan los conceptos y principios básicos de la ingeniería de requisitos. Así, se dará una visión global de los diferentes tipos de requisitos, para posteriormente presentar con detalle la notación que propone UML para la técnica de los casos de uso.

Descriptorios

Ingeniería de requisitos; requisito; restricción; obtención (elicitación) de requisitos; análisis de requisitos; especificación de requisitos *software* (ERS); modelo de casos de uso; caso de uso; actor; relaciones entre casos de uso; especificación de casos de uso.

Competencias

CC1; CC2; CC8; IS2; IS4; TI1; CT1; CT2; CT3; CT16; CT18.

Contenidos

1. Introducción.
2. Ingeniería de requisitos.
3. Requisitos.
4. Especificación de requisitos del *software*.
5. MDB: Una metodología de elicitación de requisitos.
6. Vista de casos de uso en UML.

7. Caso de estudio.

Recursos*Recursos docentes:*

- Ingeniería de requisitos [228].

Bibliografía:

1. G. Booch, J. Rumbaugh y I. Jacobson, 2ª, Ed. *El lenguaje unificado de modelado* (Object Technology Series). Madrid, España: Pearson Educación, 2007 [229] (2ª edición en inglés [130]). **Capítulo 16.**
2. A. Durán y B. Bernárdez, “Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.3),” Universidad de Sevilla, Universidad de Sevilla, España, Informe Técnico LSI-2000-10, 2002. Disponible en: <https://goo.gl/rhV8eV> [230].
3. C. Larman, UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al Proceso Unificado, 2ª ed. Madrid, España: Pearson Educación, 2003 [205] (3ª edición en inglés [206]). **Capítulos 4, 5, 6 y 7.**
4. Object Management Group, “Unified Modeling Language specification version 2.5.1,” Object Management Group, Needham, MA, USA, formal/17-12-05, 2017. Disponible en: <https://goo.gl/kaE82a> [132].
5. R. S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 7ª ed. México D. F., México: McGraw-Hill, 2010 [208] (8ª edición en inglés [5]). **Capítulo 5.**
6. J. Rumbaugh, I. Jacobson y G. Booch, *El Lenguaje Unificado de Modelado manual de referencia*, 2ª ed. (Object Technology Series). Madrid, España: Pearson Educación, 2007 [231] (2ª edición en inglés [131]).
7. I. Sommerville, *Ingeniería del Software*, 9ª ed. México: Pearson Educación, 2011 [2] (10ª edición en inglés [32]). **Capítulo 4.**

Lecturas complementarias:

1. A. Casamayor, D. Godoy y M. Campo, “Identification of non-functional requirements in textual specifications: A semi-supervised learning approach,” *Information and Software Technology*, vol. 52, no. 4, pp. 436-445, 2010. doi: 10.1016/j.infsof.2009.10.010 [232].

Artículo que aboga por la detección temprana de los requisitos no funcionales utilizando técnicas automáticas.

2. C. Ebert, “Putting requirement management into praxis: Dealing with nonfunctional requirements,” *Information and Software Technology*, vol. 40, no. 3, pp. 175-185, 1998. doi: 10.1016/S0950-5849(98)00049-4 [233].

Artículo que se centra en la gestión práctica de los requisitos no funcionales.

3. D. J. Grimshaw y G. W. Draper, “Non-functional requirements analysis: deficiencias in structured methods,” *Information and Software Technology*, vol. 43, no. 11, pp. 629-634, 2001. doi: 10.1016/S0950-5849(01)00171-9 [234].

Artículo que examina las deficiencias de los métodos estructurados a la hora de gestionar los requisitos no funcionales.

4. A. M. Hickey y A. M. Davis, “The Role of Requirements Elicitation Techniques in Achieving Software Quality,” presentado en Eighth International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ’2002 (September 09-10th, 2002), Essen, Germany, 2002 [235].

Artículo que pone de manifiesto la relación existente entre las técnicas de obtención de requisitos y la calidad del producto resultante.

5. L. A. Maciaszek, *Requirements analysis and system design: Developing information systems with UML*. Essex, UK: Addison-Wesley Longman Ltd., 2001 [236].

Cabe destacar los capítulos 3 *Requirements Determination* y 4 *Requirements Specification*.

6. P.-W. Ng, “Adopting use cases, Part 1: Understanding types of use cases and artifacts,” *IBM developerWorks*: IBM, 2003, Disponible en: <https://goo.gl/2MdMkP> [237].

Artículo que expone los diferentes tipos de casos de uso y artefactos *software* relacionados.

7. N. Power, “Variety and quality in requirements documentation,” presentado en Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ’2001 (June 4-5, 2001), Interlaken, Switzerland, 2001 [238].

Artículo que discute diferentes formas en las que los documentos de requisitos varían sus contenidos, sus propósitos o sus formas de uso.

7.6.4.5. Tema 5 – Introducción al Proceso Unificado

Resumen

En este tema se hace una presentación del Proceso Unificado. Se hace especial hincapié en sus características, su ciclo de vida y sus artefactos. En el tema siguiente se da continuidad a estas características con la descripción de los flujos de trabajo de este proceso.

Descriptorios

Proceso; Proceso Unificado; ciclo de vida; casos de uso; arquitectura *software*; iteratividad; incremental.

Competencias

CC2; CC16; TI1; CT1; CT2; CT4; CT18; CT19; CT22.

Contenidos

1. Introducción.
2. La vida del Proceso Unificado.
3. El producto.
4. El proceso.

Recursos

Recursos docentes:

- Introducción al Proceso Unificado [239].

Bibliografía:

1. I. Jacobson, G. Booch y J. Rumbaugh, *El Proceso Unificado de desarrollo de software* (Object Technology Series). Madrid, España: Pearson Educación, 2000 [240] (edición en inglés [62]). **Capítulos 1, 2, 3, 4 y 5.**

Lecturas complementarias:

1. P. B. Kruchten, “The 4+1 View Model of architecture,” *IEEE Software*, vol. 12, no. 6, pp. 42-50, 1995. doi: 10.1109/52.469759 [241].

En este artículo se presenta el patrón arquitectónico 4+1 vistas.

2. Rational Software, “Rational Unified Process. Best practices for software development teams,” Rational Software, Cupertino, CA, USA, Rational Software

White Paper, TP026B, Rev 11/01, 1998. Disponible en: <https://goo.gl/5KNng4> [242].

Buenas prácticas con el Proceso Unificado de Rational.

7.6.4.6. Tema 6 – Flujos de trabajo del Proceso Unificado

Resumen

Este tema recoge los flujos de trabajo del Proceso Unificado vinculados con los requisitos, el análisis y el diseño. Se pretende que este tema sea una referencia a estos flujos de trabajo, si bien estos mismos se van a desarrollar en los temas que profundizan sobre los conceptos ligados a estas fases del ciclo de vida desde un enfoque orientado a objetos.

Descriptorios

Proceso; Proceso Unificado; ciclo de vida; requisito; caso de uso; escenario; modelo de dominio; modelo de análisis; modelo de diseño; arquitectura *software*; clase; interfaz.

Competencias

CC2; CC16; TI1; CT1; CT2; CT4; CT18; CT19; CT22.

Contenidos

1. Requisitos en el Proceso Unificado.
2. Análisis en el Proceso Unificado.
3. Diseño en el Proceso Unificado.

Recursos

Recursos docentes:

- Flujos de trabajo del Proceso Unificado [243].

Bibliografía:

1. I. Jacobson, G. Booch y J. Rumbaugh, *El Proceso Unificado de desarrollo de software* (Object Technology Series). Madrid, España: Pearson Educación, 2000 [240] (edición en inglés [62]). **Capítulos 6, 7, 8 y 9.**

7.6.4.7. Tema 7 – Análisis orientado a objetos

Resumen

El análisis orientado a objetos consiste en una serie de técnicas y actividades mediante las que los requisitos identificados en la fase de elicitación son analizados, refinados y estructurados. El objetivo es una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema. El resultado consistirá en un modelo del sistema, modelo objeto, que describa el dominio del problema y que deberá ser correcto, completo, consistente y verificable.

Descriptores

Análisis orientado a objetos; modelo de dominio; clase conceptual; Proceso Unificado; objeto de entidad; objeto de interfaz; objeto de control.

Competencias

CC2; CC8; IS2; IS4; CT1; CT2; CT3; CT9; CT16; CT18.

Contenidos

1. Introducción.
2. Análisis orientado a objetos.
3. Modelo del dominio.
4. Requisitos en el Proceso Unificado.
5. Análisis en el Proceso Unificado.

Recursos

Recursos docentes:

- Análisis orientado a objetos [244].

Bibliografía:

1. G. Booch, J. Rumbaugh y I. Jacobson, 2ª, Ed. *El lenguaje unificado de modelado* (Object Technology Series). Madrid, España: Pearson Educación, 2007 [229] (2ª edición en inglés [130]). **Capítulos 8, 9, 10, 11, 12, 14, 15 y 18.**
2. I. Jacobson, G. Booch y J. Rumbaugh, *El Proceso Unificado de desarrollo de software* (Object Technology Series). Madrid, España: Pearson Educación, 2000 [240] (edición en inglés [62]). **Capítulo 8.**

3. C. Larman, UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al Proceso Unificado, 2ª ed. Madrid, España: Pearson Educación, 2003 [205] (3ª edición en inglés [206]). **Capítulos 9, 10, 11, 12, 26 y 27.**
4. I. Sommerville, *Ingeniería del Software*, 9ª ed. México: Pearson Educación, 2011 [2] (10ª edición en inglés [32]). **Capítulo 5.**

Lecturas complementarias:

1. B. Bruegge y A. H. Dutoit, *Object-oriented software engineering. Using UML, patterns, and Java*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010 [245].

Es interesante el capítulo 5 *Analysis*.

2. L. A. Maciaszek, *Requirements analysis and system design: Developing information systems with UML*. Essex, UK: Addison-Wesley Longman Ltd., 2001 [236].

Cabe destacar los capítulos 4 *Requirements Specification* y 5 *Advanced Analysis*.

3. Ministerio de las Administraciones Públicas, *Métrica v3*, Madrid, España: Ministerio de las Administraciones Públicas, 2001. [Online]. Disponible en: <https://goo.gl/FZ3aX4> [93].

Es interesante destacar la parte de análisis orientado a objetos de esta metodología.

4. J. J. Odell, *Advanced object-oriented analysis and design using UML* (SIGS Reference Library). SIGS Books & Multimedia, 1998 [246].

Colección de artículos relacionados con el modelado de objetos.

5. J. Rumbaugh, *OMT insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*. New York, NY, USA: SIGS Books Publications, 1996 [106].

Colección de artículos relacionados con el modelado de objetos.

6. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy y W. Lorensen, *Modelado y diseño orientados a objetos. Metodología OMT*. Madrid, España: Prentice-Hall, 1996 [247] (edición en inglés [72]).

De este libro clásico cabe destacar los capítulos 1, 3, 4, 7 y 8.

7.6.4.8. Tema 8 – UML. Unified Modeling Language

Resumen

Este es un tema de referencia y consulta. En él se presentan las diferentes vistas del Lenguaje Unificado de Modelado UML: Vista estática, Vista de gestión del modelo, Vista de casos de uso, Vista de interacción, Vista de actividad, Vista de máquina de estados, Vista de diseño, Vista de despliegue. Es un contenido de referencia común a las asignaturas *Ingeniería de Software I* e *Ingeniería de Software II*. En el caso de *Ingeniería de Software I* se pone el énfasis en la Vista estática, la Vista de casos de uso y la Vista de interacción.

Descriptorios

UML; Vista estática; Vista de gestión del modelo; Vista de casos de uso; Vista de interacción; Vista de actividad; Vista de máquina de estados; Vista de diseño; Vista de despliegue.

Competencias

CB5; CT1; CT2; CT18.

Contenidos

1. Introducción.
2. Vista estática.
3. Vista de gestión del modelo.
4. Vista de casos de uso.
5. Vista de interacción.
6. Vista de actividad.
7. Vista de máquina de estados.
8. Vista de diseño.
9. Vista de despliegue.
10. Perfiles.

Recursos

Recursos docentes:

- UML. Unified Modeling Language [248].

Bibliografía:

1. G. Booch, J. Rumbaugh y I. Jacobson, 2ª, Ed. *El lenguaje unificado de modelado* (Object Technology Series). Madrid, España: Pearson Educación, 2007 [229] (2ª edición en inglés [130]).
2. J. Rumbaugh, I. Jacobson y G. Booch, *El Lenguaje Unificado de Modelado manual de referencia*, 2ª ed. (Object Technology Series). Madrid, España: Pearson Educación, 2007 [231] (2ª edición en inglés [131]).

7.6.5. Organización de las sesiones de clase

Tradicionalmente las clases de esta asignatura se organizaban en las sesiones de teoría, en las que se desarrollaba el temario, y en las sesiones de prácticas, en las que resolvían problemas de modelado en formato taller colaborativo [249]. En tiempo fuera de clase, los estudiantes, organizados en grupos, realizaban un trabajo final consistente en desarrollar una Especificación de Requisitos del Software (ERS) de una complejidad media.

Los resultados no eran lo satisfactorios que se buscaban, aunque se aplicaron diferentes innovaciones. Por ello, en el curso 2016-2017 se decidió plantear la asignatura con enfoque activo [250] en torno al trabajo final, es decir, con un enfoque de aprendizaje basado en proyectos [251]. Todas las sesiones, se planifican con una perspectiva de una fase de Inicio y una fase de Elaboración, siguiendo el Proceso Unificado, de forma que se van entregando unos hitos parciales, que, de forma iterativa e incremental, terminarán conformando el entregable final, tal y como se puede apreciar en la Figura 7.25, que muestra la planificación temporal real para el curso 2017-2018.

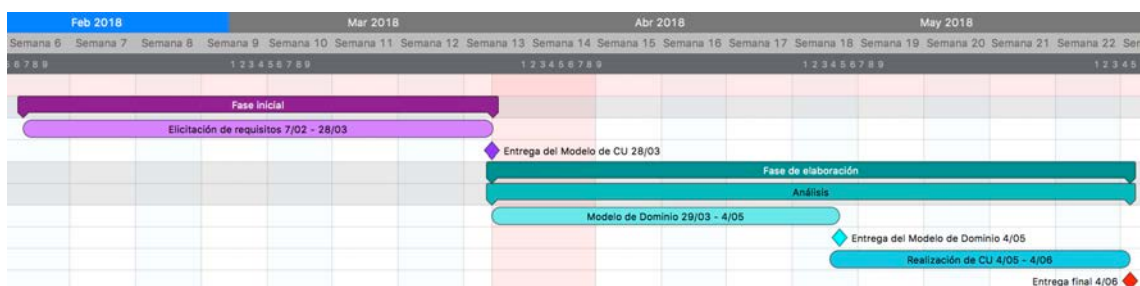


Figura 7.25. Planificación temporal de los hitos para la entrega del trabajo final. Fuente: [117] (p. 25)

Se tienen diferentes tipos de sesiones de clase para cumplir los hitos marcados, que a su vez satisfarán las competencias y los objetivos de aprendizaje de la asignatura. Concretamente se tienen:

- Sesiones de clase teórica de una hora duración (grupo completo).

- Sesiones de clase de problemas de modelado de una hora de duración (grupo completo).
- Sesiones de trabajo grupal en la práctica final de una hora de duración (grupo completo).
- Sesiones de clase práctica para presentar los fundamentos de UML de dos horas de duración (el grupo global se divide en subgrupos).
- Sesiones de resolución colaborativa de problemas en formato taller de dos horas de duración (el grupo global se divide en subgrupos).

La planificación real de las sesiones de clase de una hora (grupo completo) para el curso 2017-2018 se muestra en la Figura 7.26 (modo compacto) y en la Figura 7.27 (modo calendario).

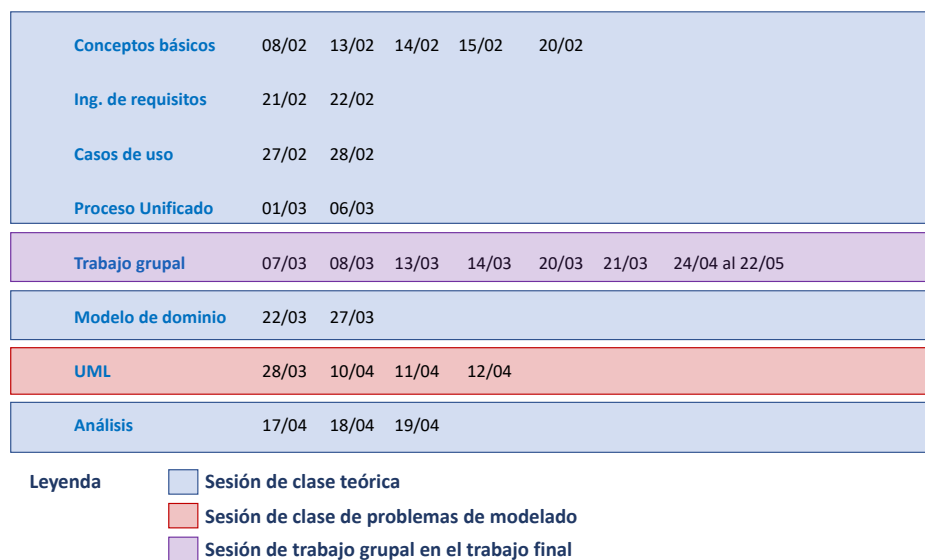


Figura 7.26. Planificación temporal de las sesiones de clase de grupo completo (formato compacto).
Fuente: Basado en [117] (p. 19)



Figura 7.27. Planificación temporal de las sesiones de clase de grupo completo (formato calendario).
Fuente: Basado en [117] (p. 20)

La planificación real de las sesiones de clase de dos horas (subgrupos) para el curso 2017-2018 se muestra en la Figura 7.28 (modo compacto) y en la Figura 7.29 (modo calendario).

	CU	T1	UML	T2	UML	T3	UML
Subgrupo A1	13/02	27/02	13/03	23/03	10/04	27/04	08/05
Subgrupo A2	08/02	22/02	08/03	22/03	12/04	26/04	10/05

Leyenda

- Sesión de fundamentos de UML
- Sesión de taller

Figura 7.28. Planificación temporal de las sesiones de clase de los subgrupos (formato compacto).
Fuente: Basado en [117] (p. 21)



Figura 7.29. Planificación temporal de las sesiones de clase de los subgrupos (formato calendario).
Fuente: Basado en [117] (p. 22)

7.6.5.1. Clases de teoría y de fundamentos de UML

El temario completo de la asignatura está disponible desde que comienza la asignatura en el espacio de la misma en el campus virtual institucional de la Universidad de Salamanca, Studium (ver Figura 7.30). En el enfoque activo que se sigue, este temario es una fuente de referencia para los estudiantes, pero el objetivo no será impartirlo por completo, sino seleccionar los conceptos necesarios para poder ir avanzando en el desarrollo del trabajo final. Para ello se ha hecho una selección de contenidos, así como

del orden en que se imparten, para las sesiones de teoría, de una hora de duración, y de fundamentos de UML, de dos horas de duración.

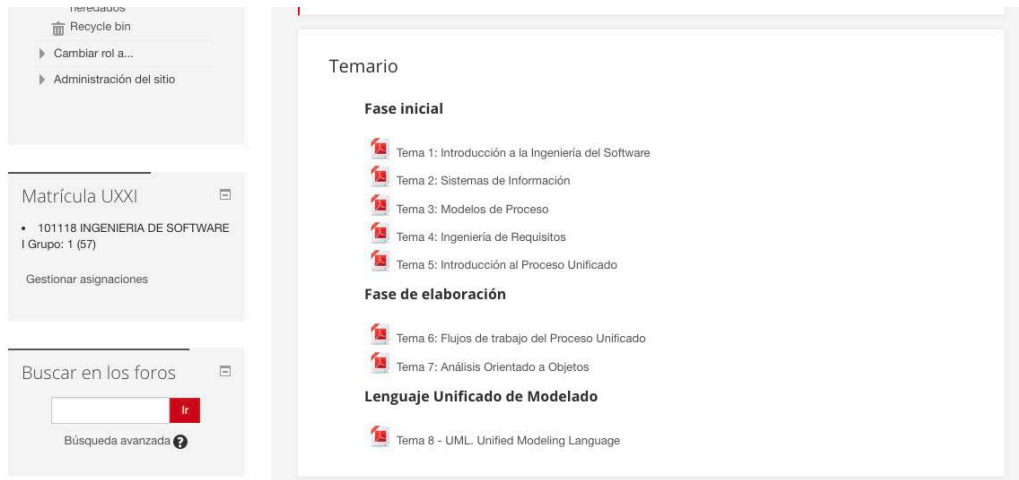


Figura 7.30. Captura del campus virtual del bloque con los temas de la asignatura

Concretamente, se tienen 17 sesiones de teoría (donde se incluye la sesión de presentación de la asignatura) y 4 sesiones de fundamentos de UML.

Las sesiones de clase de teoría de una hora de duración, que se imparten al grupo completo, se organizan de la siguiente manera, conforme a la planificación temporal que se muestra en las figuras 26 y 27:

- Bloque de presentación:
 - Sesión 0: Presentación de la asignatura [117] (1 hora).
- Bloque de conceptos básicos:
 - Sesión 1: Concepto de *software* [252] y de Ingeniería del Software [253] (1 hora).
 - Sesión 2 y Sesión 3: Proceso [254] (2 horas).
 - Sesión 4 y Sesión 5: Metodologías de Ingeniería de Software [255] (2 horas).
- Bloque de ingeniería de requisitos:
 - Sesión 6 y Sesión 7: Requisitos [256] (2 horas).
 - Sesión 8 y Sesión 9: Aspectos prácticos de los casos de uso [257] (2 horas).
- Bloque de Proceso Unificado:
 - Sesión 10 y Sesión 11: Proceso Unificado [258] (2 horas).
- Bloque de análisis orientado a objetos:
 - Sesión 12 y Sesión 13: Modelo de dominio [259] (2 horas).

- Sesión 14, Sesión 15 y Sesión 16: Introducción al análisis orientado a objetos [260] (3 horas).

Las sesiones de fundamentos de UML de dos horas de duración, que se imparten a cada uno de los subgrupos, se organizan de la siguiente manera conforme a la planificación temporal que se muestra en las figuras 28 y 29:

- Sesión 1: Fundamentos de la vista de casos de uso [261] (2 Horas).
- Sesión 2 y Sesión 3: Fundamentos de la vista estática [262] (4 Horas).
- Sesión 4: Fundamentos de la vista de interacción [263] (2 Horas).

Todos estos recursos están accesibles en el campus virtual, organizados para apoyar los tres hitos de entrega del trabajo final, como se presentó en la Figura 7.25. La mayor parte de las sesiones de clases teóricas están programadas en la parte inicial de la asignatura, dentro de la *Fase de Inicio: Elicitación de requisitos*, como se recoge en la Figura 7.31. El resto de los contenidos se reparten en la *Fase de Elaboración: Modelo de dominio* (ver Figura 7.32) y en la *Fase de Elaboración: Realización de casos de uso* (ver Figura 7.33).

Fase de inicio: elicitación de requisitos

Soporte teórico

- Concepto de software
Se define y categoriza el concepto de software
- Concepto de Ingeniería del Software
Se define qué es Ingeniería del Software, se introduce el método de Ingeniería del Software y la diferencia entre Dominio del Problema y el Dominio de la Solución
- Proceso
Proceso Software. Principales modelos de proceso
- Metodologías de Ingeniería de Software
Concepto de metodologías software. Metodologías estructuradas vs. metodologías orientadas a objetos. Metodologías Ágiles
- Requisitos
Proceso de Ingeniería de Requisitos; Requisito; Especificación de Requisitos del Software
- Aspectos prácticos de los casos de uso
Recomendaciones sobre los casos de uso. Desarrollo de un ejemplo
- Proceso Unificado
Proceso Unificado; Ciclo de Vida

Soporte práctico

- Fundamentos de la vista de casos de uso
Se introduce el lenguaje unificado de modelado (UML) y se presenta la vista de casos de uso.

Figura 7.31. Captura del campus virtual del bloque con los recursos asociados a la *Fase de Inicio: Elicitación de requisitos*

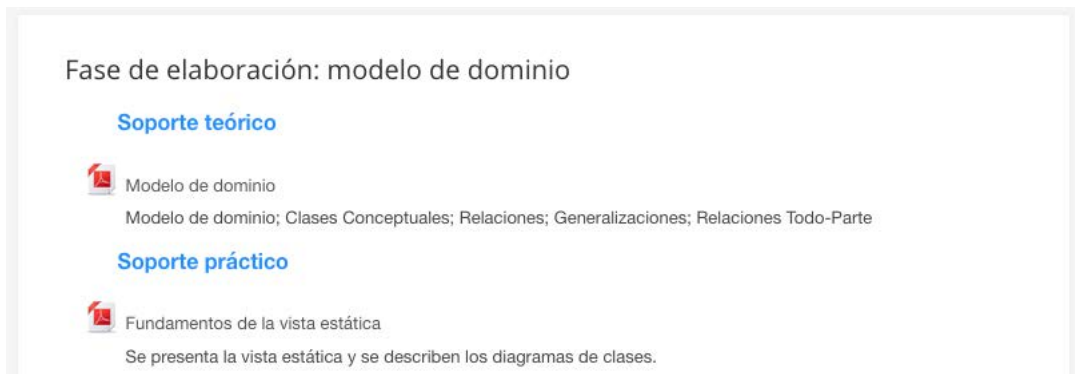


Figura 7.32. Captura del campus virtual del bloque con los recursos asociados a la *Fase de Elaboración: Modelo de dominio*

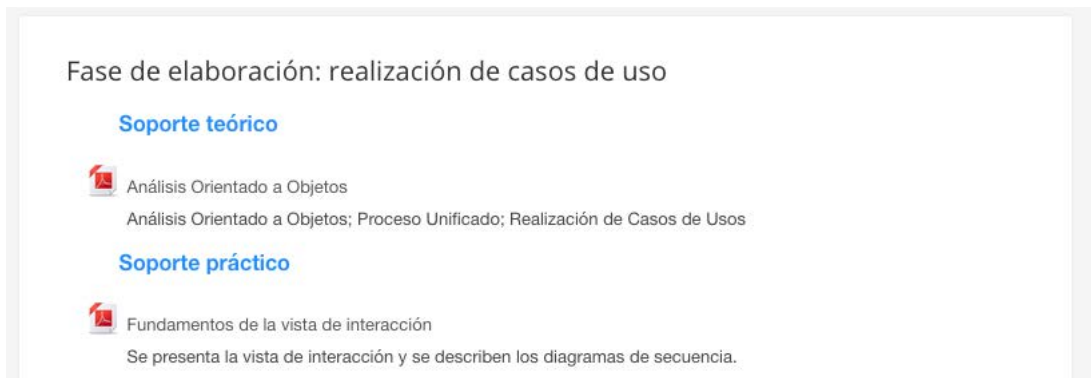


Figura 7.33. Captura del campus virtual del bloque con los recursos asociados a la *Fase de Elaboración: Realización de casos de uso*

7.6.5.2. Clases de problemas

El objetivo de aprendizaje “Modelar un sistema *software* en diferentes niveles de abstracción mediante el uso de un lenguaje de modelado estándar” es fundamental en esta asignatura. El modelo de dominio de un sistema es clave para la documentación de los requisitos y la continuidad del ciclo de vida.

Los conceptos teóricos se deben llevar a la práctica, las clases de talleres son muy importantes para este objetivo y se realizan de forma grupal y se discuten con el subgrupo para aprender de las buenas prácticas y también de los errores. Sin embargo, también es necesario enfrentarse a esta tarea de forma individual para desarrollar la competencia de resolución de problemas.

Se han programado cuatro sesiones de resolución de problemas, para lo que se facilitará un conjunto de enunciados de problemas, de forma que los estudiantes los puedan resolver individualmente y se discutirán en estas sesiones.

Las competencias asociadas a estas sesiones son: IS4; CT1; CT2; CT3; CT5; CT9; CT10; CT11; CT16; CT18; CT20; CT21.

7.6.5.3. Sesiones de trabajo grupal en la práctica final

Con el cambio metodológico hacia un enfoque activo, el trabajo final se ha convertido en el hilo conductor de la asignatura. Todos los contenidos teóricos seleccionados se imparten con una planificación temporal para que sirvan de sustento al trabajo práctico. De una orientación completamente autónoma de los grupos para hacer este trabajo, fuera del horario de clases teóricas y prácticas, se le ha hecho un hueco en las horas presenciales, no para eliminar todo el trabajo autónomo de los grupos, pero sí para que haya un espacio de interacción con los docentes para comentar los avances y, también, obligar a que el desarrollo de este trabajo esté planificado y comience desde el principio de la asignatura.

Es decir, con este cambio y la incorporación de esta tipología de sesiones presenciales se ha pasado de una asignatura centrada en los contenidos, que se veían reflejados –no siempre con éxito en la práctica–, a una asignatura centrada en las competencias profesionales del *saber hacer* para lo cual *necesitan conocer* y van a tener que demostrarlo con el resultado de un trabajo, que es a la vez autónomo y supervisado, en el que las entregas parciales sirven de retroalimentación para el siguiente incremento, en el que además de los nuevos objetivos planificados se pueden (se deben) incorporar los comentarios que posibilitarán no solo mejorar el producto, aprendiendo de los errores, sino también ver reflejadas las mejoras realizadas en la nota final de este trabajo.

7.6.5.3. Talleres

Las sesiones de resolución colaborativa de problemas en formato taller constituyen una parte importante de las horas de práctica de la asignatura y se llevan impartiendo desde el curso 1998-1999 en las diferentes asignaturas de los distintos planes de estudio.

En ellos la participación activa, la interacción y la discusión entre los estudiantes que participan en el taller, con la mediación e intervención justa del docente, es de capital importancia para el éxito del taller. Si no se da esta participación activa el taller pierde su sentido y no sería más que otra clase de resolución de problemas como las que ya se han comentado previamente.

Los objetivos generales de estos talleres se pueden resumir en los siguientes puntos:

- Practicar el modelado de sistemas *software* usando el paradigma objetual y UML.
- Comentar en público los errores más frecuentes que se cometen a la hora de realizar los modelos.

- Potenciar la comunicación oral en público de los estudiantes.
- Introducir a la realización de informes técnicos.
- Incentivar la participación activa del alumnado en el desarrollo de la asignatura.

Como se puede apreciar en la Figura 7.28, se han programado tres talleres, de dos horas de duración cada uno, a los que asistirán los estudiantes organizados en subgrupos para facilitar así la interacción y el debate al ser un número más reducida de asistentes. Como todas las demás sesiones de clase, los talleres también se han programado y sincronizado con las clases teóricas y de fundamentos de UML para que sean efectivos desde el punto de vista del trabajo final. Así, se comienza con taller de modelado de casos de uso y después se tienen programados dos talleres de modelado de clases.

El taller se considera una actividad evaluable que se computará en el apartado de evaluación continua de la asignatura.


Para el desarrollo de los talleres no se requiere un aula diferente a la de teoría. Los estudiantes se organizan en grupos de tres personas, que deben ser los mismos componentes que están desarrollando el trabajo final de la asignatura.


Cada taller, para cada uno de los subgrupos, quedará completamente definido previamente a su desarrollo en el campus virtual de la asignatura, con un enunciado y entrega de tarea que se cierra antes de la celebración del taller, como queda reflejado en la Figura 7.34. Cada grupo que quiera participar en la evaluación continua debe entregar previamente su propuesta de solución, es suficiente con un boceto. Esta solución previa no puntúa, pero si se detecta fraude en su entrega se resta un punto en la evaluación continua.


Taller de modelado de Requisitos

En este taller se deberá realizar un modelo de casos de uso utilizando como referencia el enunciado que aparece a continuación.

Exclusivamente el portavoz del grupo entregará, a través de esta plataforma y con anterioridad al desarrollo del taller, un documento PDF con los miembros del grupo y la solución propuesta.

 Taller Casos de Uso - Grupo A2
Este enunciado es para el grupo de prácticas A2 y se debe entregar antes de la clase de prácticas del jueves 22 de febrero.

 Entrega taller Casos de Uso - Grupo A2

 Taller Casos de Uso - Grupo A1
Este enunciado es para el grupo de prácticas A1 y se debe entregar antes de la clase de prácticas del martes 27 de febrero.


 Entrega taller Casos de Uso - Grupo A1

Figura 7.34. Captura del campus virtual con la definición del taller de modelado de casos de uso para los dos subgrupos A1 y A2 del curso 2017-2018

Al comienzo del taller, se solicita un grupo voluntario para que presente su solución al resto de sus compañeros (si hubiera más de un grupo se sortea y si no hubiera grupo voluntario se cancela el taller).

En la pizarra el grupo voluntario dibuja su solución y la exponen a sus compañeros. Una vez que han terminado se exposición comienza el debate grupal moderado por el profesor o profesores. Cada grupo en base a su propia solución y la solución presentada opinan sobre errores, variantes, cambios, etc. Se deja que sea el grupo ponente el que haga una defensa de su propuesta, siempre razonando el porqué de si acepta o desestima el comentario. El profesor podrá incorporar las alternativas propuestas y cerrará cada discusión con su valoración.

Finalmente, el grupo voluntario puede realizar un informe con la solución final alcanzada con entrega en 15 días tras el taller y que se comparte a todos los subgrupos para que haya más material de referencia.

Toda la participación en el taller se puntúa y contribuye a la nota de evaluación continua.

- Por la defensa el grupo voluntario obtiene entre 0 y 0,75 puntos.
- Por la entrega del informe el grupo voluntario obtiene hasta 0,75 puntos.
- La participación activa, acertada y continuada en el debate de todos los talleres puede aportar hasta 0,5 puntos de forma individual.

Las competencias asociadas a este tipo de sesiones son: CC1; IS2; IS4; CT1; CT2; CT3; CT5; CT9; CT10; CT11; CT12; CT16; CT18; CT20; CT21.

7.6.6. Práctica final obligatoria

Ya se ha venido comentando la importancia que toma la práctica o trabajo final de la asignatura en el enfoque activo que se le ha dado a la misma. Este trabajo ha pasado de ser un producto evaluable como receptor de los contenidos teóricos a ser el hilo conductor de la asignatura y, aunque no el único, el motor de la participación activa de los estudiantes.

En las primeras semanas de la asignatura queda publicado en el campus virtual la metodología de trabajo, que se presentó en la [Figura 7.25](#) como argumento para planificar las sesiones de clase de la asignatura, el enunciado de la misma (este es el propuesto para el curso 2017-2018 [264]), un conjunto de indicaciones y recomendaciones para su desarrollo [265] y las rúbricas para la evaluación de cada uno

de los hitos que conforman la entrega incremental de la práctica. Todo ello se puede apreciar en la Figura 7.35.

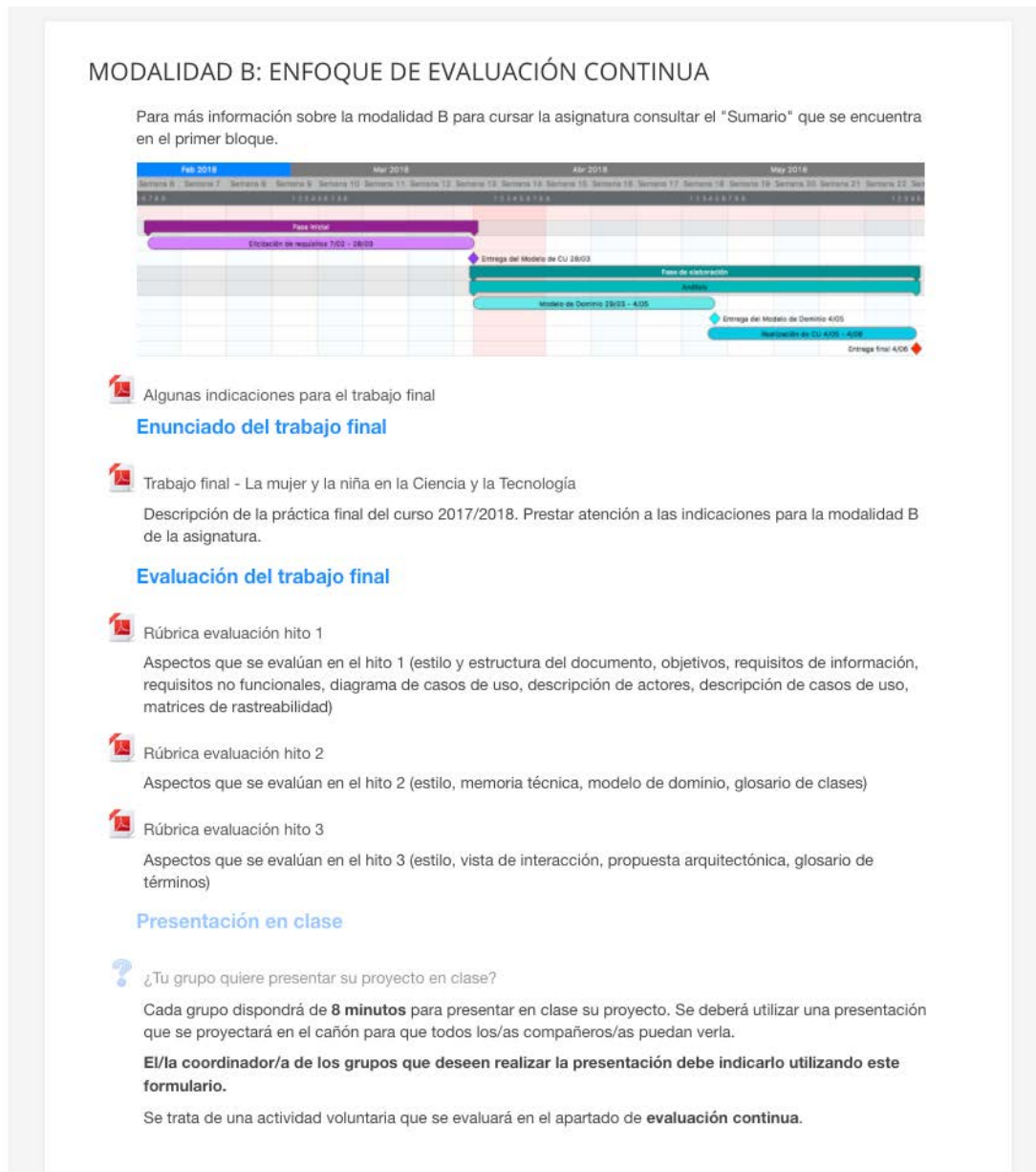


Figura 7.35. Captura del campus virtual con la información del trabajo final de la asignatura (orientada a la modalidad de evaluación continua). Curso 2017-2018

El trabajo práctico se orienta desde la perspectiva de enfrentar al estudiante a la problemática derivada de tener que afrontar el desarrollo de un producto *software* basado en unos requisitos reales.

El trabajo abarca la fase de obtención y especificación de requisitos y la fase de análisis de los mismos.

La forma de entrega difiere dependiendo de la modalidad elegida para cursar la asignatura (ver [apartado 7.6.7](#)). En el caso de la modalidad enfocada hacia una

evaluación final, se entregará una memoria en formato digital (a través de la tarea habilitada para ello en el campus virtual). En el caso de la modalidad de evaluación continua, se realizarán dos entregas parciales obligatorias a través de una carpeta compartida en Google Drive USAL y una entrega final a través de la tarea habilitada para ello en el campus virtual.

Independientemente de la modalidad elegida, la memoria final constará de una estructura detallada, pero que en esencia incluirán una introducción, los objetivos, las técnicas y herramientas, la descripción del grupo de trabajo, los aspectos relevantes, las conclusiones y la documentación técnica compuesta por el catálogo de requisitos que busca satisfacer (documentación de requisitos), su especificación y el modelo de análisis.

La práctica se realizará en grupos de tres personas (salvo excepciones justificadas y que serán las mismas personas que trabajan en los talleres) que cursen la misma modalidad de la asignatura. Una de las personas del grupo tomará el rol de jefe de equipo y se encargará de coordinar las tareas dentro de su grupo. El grupo completo será responsable de las actividades de sus miembros, esto es, aunque haya una división de tareas dentro del grupo, debe existir una comunicación dentro del grupo de forma que todos los implicados estén al tanto de las actividades del resto, existiendo una coordinación entre las actividades.

Cada curso académico se propone un tema monográfico sobre el que los integrantes de cada grupo deben investigar y desarrollar una propuesta donde la creatividad será recompensada. Concretamente, en el curso académico 2017-2018 la propuesta es modelar una aplicación (web o *app* para teléfono móvil) cuyo tema central sea la mujer y la niña en la Ciencia y la Tecnología [264] con el fin de reducir la brecha de género en el ámbito científico y tecnológico. La funcionalidad de la herramienta no debe reducirse a recopilar y mostrar información relacionada con la temática, sino que debe enfocarse en trabajar algún aspecto relacionado con la misma. Se continua así con la concienciación sobre los aspectos de género en la profesión de ingeniero en informática, que se comenzó en el curso 2016-2017 con el proyecto de innovación ID2016/084 [266], del que se puede conocer más en los trabajos [267-269].

Esta forma de plantear la práctica obligatoria tiene las siguientes ventajas:

- Se obliga a que los estudiantes se acerquen a una perspectiva real [270].
- Evita el plagio de prácticas dado que cada grupo su propio enfoque.

- Potencia el trabajo en grupo. Se les da libertad para que ellos se organicen.
- Se potencia el trabajo autónomo, pero también la realimentación en las clases de trabajo grupal.
- Se les da un marco de planificación basado en el Proceso Unificado [62], pero se les recomienda una aproximación ágil [64] a las tareas, lo que transmite un enfoque híbrido [168] para afrontar el desarrollo de los proyectos aprovechando lo mejor de cada enfoque y ayuda a huir de los extremismos tecnológicos.
- Se hace hincapié en la utilización de estándares para realizar los documentos entregables, aunque se les da libertad para configurar su entorno de trabajo tecnológicos, tanto en cuanto a herramientas CASE como a procesadores de texto, aunque se les ofrece una variada selección de opciones en el campus virtual, como se puede apreciar en la Figura 7.36.

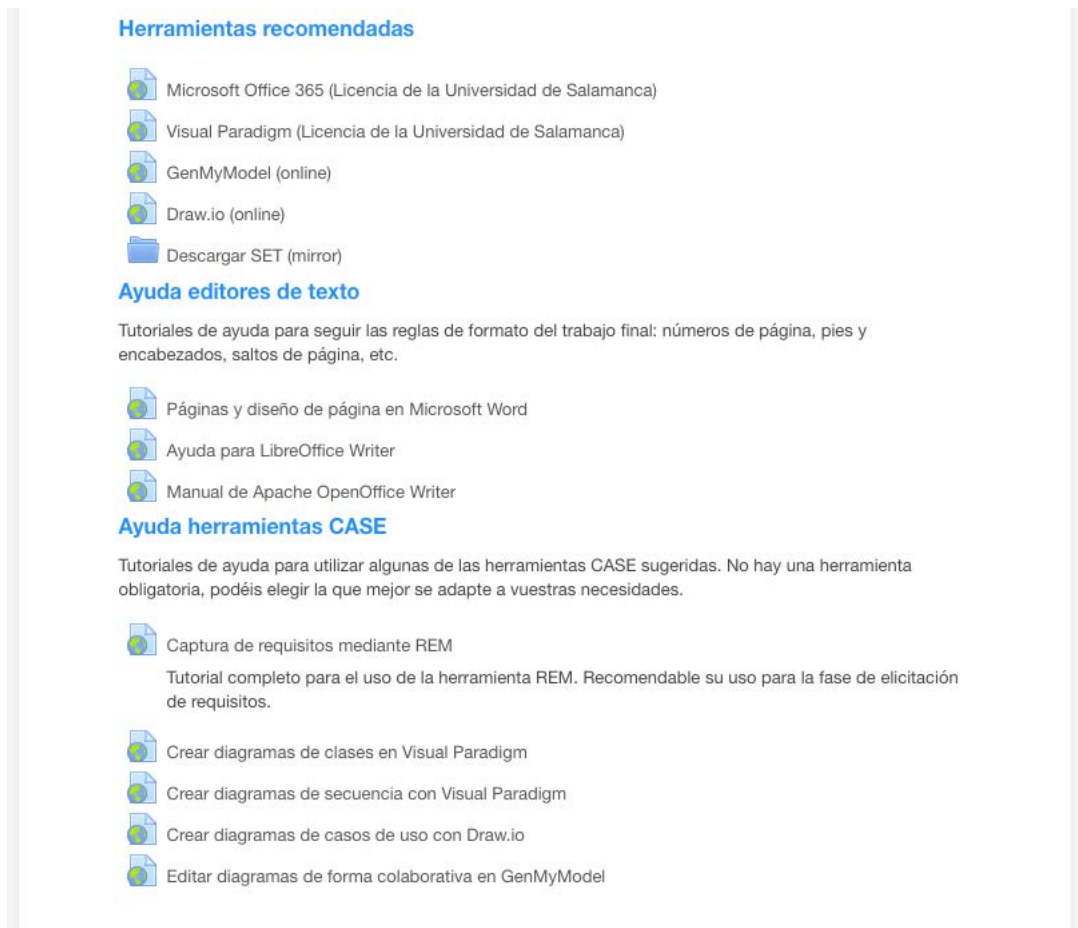


Figura 7.36. Captura del campus virtual con las recomendaciones sobre qué herramientas usar a la hora de realizar la práctica obligatoria y los informes de los talleres

Como se ha indicado la entrega del trabajo realizado depende de la modalidad en la que se curse la asignatura.

En la modalidad de evaluación continua (ver Figura 7.35), los integrantes de un grupo se comprometen a asistir a las clases, con especial énfasis a aquellas en que se trabaja en grupo en el trabajo final con la supervisión del equipo docente. En este caso se hacen dos entregas parciales en una carpeta compartida en Google Drive USAL y una entrega final a través del campus virtual. El grupo deberá quedarse con copia del material entregado porque este no le será devuelto (ni prestado). Tampoco se admitirán modificaciones en los artefactos después de la fecha límite. Además, una vez finalizado el curso se retirará el acceso a las carpetas compartidas en Google Drive.

El primer hito, que se corresponde con la primera entrega parcial, debe tener la estructura del documento e incluir el catálogo de requisitos junto con todos los ficheros asociados (modelos, etc.). Se recomienda un máximo de 10 casos de uso no triviales – casos de uso que no sean CRUD (*Create, Retrieve, Update, Delete* – Crear, Recuperar, Modificar, Eliminar) – un número superior no supondrá mayor nota y lo que supondrá es un mayor esfuerzo a los integrantes del grupo. La rúbrica para evaluar el primer hito, accesible en el campus virtual para que todos los estudiantes sepan cómo se evalúa la práctica, se presenta en la Tabla 7.5.

Tabla 7.5. Rúbrica para evaluar el primer hito del trabajo final

	Insuficiente (0)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Portada	No tiene portada	No aparecen todos los datos (título, subtítulo, versión, fecha, autores) ni cumple el estilo definido	Aparecen todos los datos, pero no cumple el estilo definido	Aparecen todos los datos y cumple el estilo definido	5%	
Tabla de contenidos	No tiene tabla de contenidos	Tiene tabla de contenidos, pero no cumple el estilo definido ni se ha generado automáticamente	Tiene tabla de contenidos que cumple el estilo definido, pero no se ha generado automáticamente	Tiene tabla de contenidos generada automáticamente y cumple el estilo definido	5%	
Estilo del documento	No cumple el estilo definido para las páginas de contenido	Los encabezados no cumplen el estilo definido para las páginas de contenido	Cumple el estilo definido para las páginas de contenido, pero no se han introducido saltos de página	Cumple el estilo definido para las páginas de contenido e incluye los saltos de página	5%	
Objetivos	No se han definido los objetivos relacionados con la funcionalidad	Se han definido los objetivos, pero no son correctos	Se han definido correctamente los objetivos	Destacan por su originalidad	10%	
Requisitos de información	No se han definido los requisitos de información	Se han definido los requisitos de información, pero no se han descrito correctamente	Se han descrito correctamente los requisitos de información, pero falta información que se menciona en los casos de uso	Se han descrito correctamente todos los requisitos de información	10%	
Requisitos no funcionales	No se han definido los requisitos no funcionales	Se han definido requisitos no funcionales, pero no son correctos	Se han definido correctamente entre 1-3 requisitos no funcionales	Se han descrito correctamente más de 3 requisitos no funcionales	10%	

	Insuficiente (0)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Diagrama de casos de uso	No se ha realizado el diagrama de casos de uso	Se ha realizado el diagrama, pero no se ha utilizado bien la notación	Se ha utilizado bien la notación, pero no se han definido bien todos los casos de uso	Se han definido correctamente los casos de uso y se ha utilizado correctamente la notación	10%	
Descripción de actores	No se han descrito los actores	No hay diagrama de actores y el problema lo requiere o el diagrama de actores no es correcto	El diagrama de actores no es óptimo	Se han descrito correctamente todos los actores	10%	
Descripción de casos de uso	No se han descrito los casos de uso	Se han descrito los casos de uso, pero los pasos del escenario principal no están bien descritos o los casos de uso no se corresponden con el diagrama	Se han descrito los casos de uso, pero no se han definido excepciones, precondiciones o post-condiciones en ninguno de los casos de uso	Se han descrito correctamente todos los casos de uso y se corresponden con el diagrama	25%	
Matriz de rastreabilidad: obj-req	No se ha realizado la matriz de objetivos con requisitos	Se ha realizado la matriz, pero no es correcta	Se ha realizado la matriz, pero no todos los requisitos están asociados a un objetivo	Se ha realizado la matriz de objetivos con requisitos correctamente	5%	
Matriz de rastreabilidad: req-req	No se ha realizado la matriz de requisitos con requisitos	Se ha realizado la matriz, pero no es correcta	Se ha realizado la matriz, pero no aparecen todos los requisitos	Se ha realizado la matriz de requisitos con requisitos correctamente	5%	
					TOTAL	

El segundo hito, que se corresponde con la segunda entrega parcial que debe contener el modelo de dominio y una primera versión del documento de descripción, es decir, un documento con una extensión mínima de 5 páginas con las secciones: Introducción, objetivos, técnicas y herramientas, descripción del grupo de trabajo, aspectos relevantes y conclusiones (a semejanza de la estructura de un trabajo de fin de grado, con el objetivo de que desde un momento temprano en los estudios, los estudiantes se vayan acostumbrando a la redacción de documentación técnica). La rúbrica para evaluar el segundo hito, igualmente accesible en el campus virtual, se presenta en la Tabla 7.6.

Tabla 7.6. Rúbrica para evaluar el segundo hito del trabajo final

	Insuficiente (0)	Mal (2)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Estilo del documento	No cumple el estilo definido para las páginas de contenido		Los encabezados no cumplen el estilo definido para las páginas de contenido	Cumple el estilo definido para las páginas de contenido, pero no se han introducido saltos de página	Cumple el estilo definido para las páginas de contenido e incluye los saltos de página	5%	
Memoria técnica	No se ha realizado la memoria		Se ha realizado, pero no cumple todos los criterios definidos: 5 páginas como mínimo y los 6 apartados indicados	Tiene todos los apartados, llega al mínimo de páginas, pero tiene faltas de ortografía o no está expresado correctamente	Tiene todos los apartados, llega al mínimo o supera el número de páginas y está escrito correctamente	25%	
Diagrama de clases del	No se ha realizado el	Se ha realizado, pero no desde un punto de vista	Se ha planteado bien el modelo, pero tiene	Se ha utilizado bien la notación, pero el diagrama	Se ha realizado correctamente el diagrama de	60%	

	Insuficiente (0)	Mal (2)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
modelo de dominio	diagrama de clases	conceptual o no se ha utilizado bien la notación	demasiados errores	no es del todo correcto o faltan clases y relaciones basándose en la especificación de requisitos	clases y se ha utilizado bien la notación		
Glosario de clases	No se ha realizado el glosario de clases		Se ha realizado, pero no se han descrito todas las clases o no se ha explicado su significado	Se han descrito todas las clases y se ha explicado su significado, pero no se incluye la descripción de sus principales atributos y métodos	Se han descrito todas las clases y se ha explicado correctamente su significado, atributos y métodos principales	10%	
						TOTAL	

El tercer hito se corresponde con la entrega de la versión final. La rúbrica para evaluar el tercer hito, igualmente accesible en el campus virtual, se presenta en la Tabla 7.7.

Tabla 7.7. Rúbrica para evaluar el tercer hito del trabajo final

	Insuficiente (0)	Mal (2)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Estilo del documento	No cumple el estilo definido para las páginas de contenido		Los encabezados no cumplen el estilo definido para las páginas de contenido	Cumple el estilo definido para las páginas de contenido, pero no se han introducido saltos de página	Cumple el estilo definido para las páginas de contenido e incluye los saltos de página	5%	
Vista de interacción	No se han realizado los diagramas de secuencia	No se ha utilizado bien la notación o los diagramas tienen errores muy graves	Se han realizado menos de 10 diagramas de secuencia o no se ha utilizado bien la notación	Se han realizado entre 10-15 diagramas de secuencia y todos son correctos	Se han realizado más de 15 diagramas de secuencia y todos son correctos	50%	
Propuesta de arquitectura	No se ha realizado la propuesta arquitectónica		Se ha realizado, pero es incorrecta	Se ha realizado, pero no es óptima	Se ha realizado y es óptima	40%	
Glosario	No se ha realizado el glosario			Se ha realizado, pero tiene faltas de ortografía o no está expresado correctamente	Se ha realizado y está correctamente redactado	5%	
						TOTAL	

Se podrá realiza una defensa bajo demanda del equipo docente, en cuyo caso, aunque haya habido división de tareas, todos los miembros del grupo tienen la obligación de conocer el producto completo. Además, en las últimas sesiones de trabajo grupal se podrá presentar, de forma opcional, el trabajo realizado en una sesión organizada para tal fin.


La media de las notas obtenidas en las tres entregas computará un 90% de la nota final del trabajo y el 10% restante será el desempeño evaluado por los compañeros del grupo.


En caso de no obtener la nota mínima de 5, se realizará otra entrega con las correcciones oportunas en la segunda convocatoria de la asignatura.


Para aquellos estudiantes que, teniendo que hacer el trabajo final, hayan elegido la opción enfocada a una entrega única, sin evaluación continua, tendrán que realizar una entrega única e igualmente podrán ser convocados a una defensa grupal si así lo estima oportuno el equipo docente, aunque también existe una rúbrica para su calificación accesible en el campus virtual (Figura 7.37) y que se puede ver en la Tabla 7.8. El resto de las condiciones se comparte con los estudiantes de la modalidad de evaluación continua.


MODALIDAD A: ENFOQUE DE EVALUACIÓN FINAL

Se debe leer cuidadosamente el enunciado y cumplir las restricciones que se expresan para que la práctica no sea rechazada por defecto de forma. Para más información sobre la modalidad A para cursar la asignatura consultar el "Sumario" que se encuentra en el primer bloque.

 Algunas indicaciones para el trabajo final
[Enunciado del trabajo final](#)

 Trabajo final - La mujer y la niña en la Ciencia y la Tecnología
 Descripción de la práctica final del curso 2017/2018. Prestar atención a las indicaciones correspondientes a la modalidad A de la asignatura.
[Evaluación del trabajo final](#)

 Rúbrica de evaluación
 Aspectos que se evalúan del trabajo final.
[Tarea para la entrega](#)

 Entrega trabajo final - Modalidad A
 Debéis subir aquí la memoria en formato PDF junto con todos los ficheros relacionados con la realización del trabajo final en un único archivo zip.


 SEGUNDA CONVOCATORIA: Entrega trabajo final - Modalidad A

Figura 7.37. Captura del campus virtual con la información del trabajo final de la asignatura (orientada a la modalidad de evaluación final). Curso 2017-2018

Tabla 7.8. Rúbrica para evaluar las prácticas finales no desarrolladas por evaluación continua

	Insuficiente (0)	Mal (2)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Portada	No tiene portada		No aparecen todos los datos (título, subtítulo, versión, fecha, autores) ni cumple el estilo definido	Aparecen todos los datos, pero no cumple el estilo definido	Aparecen todos los datos y cumple el estilo definido	2%	
Tabla de contenidos	No tiene tabla de contenidos		Tiene tabla de contenidos, pero no cumple el estilo definido ni se ha generado automáticamente	Tiene tabla de contenidos que cumple el estilo definido, pero no se ha generado automáticamente	Tiene tabla de contenidos generada automáticamente y cumple el estilo definido	2%	

	Insuficiente (0)	Mal (2)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Estilo del documento	No cumple el estilo definido para las páginas de contenido		Los encabezados no cumplen el estilo definido para las páginas de contenido	Cumple el estilo definido para las páginas de contenido, pero no se han introducido saltos de página	Cumple el estilo definido para las páginas de contenido e incluye los saltos de página	2%	
Objetivos	No se han definido los objetivos relacionados con la funcionalidad		Se han definido los objetivos, pero no son correctos	Se han definido correctamente los objetivos	Destacan por su originalidad	3%	
Requisitos de información	No se han definido los requisitos de información		Se han definido los requisitos de información, pero no se han descrito correctamente	Se han descrito correctamente los requisitos de información, pero falta información que se menciona en los casos de uso	Se han descrito correctamente todos los requisitos de información	3%	
Requisitos no funcionales	No se han definido los requisitos no funcionales		Se han definido requisitos no funcionales, pero no son correctos	Se han definido correctamente entre 1-3 requisitos no funcionales	Se han descrito correctamente más de 3 requisitos no funcionales	3%	
Diagrama de casos de uso	No se ha realizado el diagrama de casos de uso		Se ha realizado el diagrama, pero no se ha utilizado bien la notación	Se ha utilizado bien la notación, pero no se han definido bien todos los casos de uso	Se han definido correctamente los casos de uso y se ha utilizado correctamente la notación	4%	
Descripción de actores	No se han descrito los actores		No hay diagrama de actores y el problema lo requiere o el diagrama de actores no es correcto	El diagrama de actores no es óptimo	Se han descrito correctamente todos los actores	3%	
Descripción de casos de uso	No se han descrito los casos de uso		Se han descrito los casos de uso, pero los pasos del escenario principal no están bien descritos o los casos de uso no se corresponden con el diagrama	Se han descrito los casos de uso, pero no se han definido excepciones, precondiciones o post-condiciones en ninguno de los casos de uso	Se han descrito correctamente todos los casos de uso y se corresponden con el diagrama	9%	
Matriz de rastreabilidad: obj-req	No se ha realizado la matriz de objetivos con requisitos		Se ha realizado la matriz, pero no es correcta	Se ha realizado la matriz, pero no todos los requisitos están asociados a un objetivo	Se ha realizado la matriz de objetivos con requisitos correctamente	1%	
Matriz de rastreabilidad: req-req	No se ha realizado la matriz de requisitos con requisitos		Se ha realizado la matriz, pero no es correcta	Se ha realizado la matriz, pero no aparecen todos los requisitos	Se ha realizado la matriz de requisitos con requisitos correctamente	1%	
Memoria técnica	No se ha realizado la memoria		Se ha realizado, pero no cumple todos los criterios definidos: 5 páginas como mínimo y los 6 apartados indicados	Tiene todos los apartados, llega al mínimo de páginas, pero tiene faltas de ortografía o no está expresado correctamente	Tiene todos los apartados, llega al mínimo o supera el número de páginas y está escrito correctamente	8%	
Diagrama de clases del modelo de dominio	No se ha realizado el diagrama de clases	Se ha realizado, pero no desde un punto de vista conceptual o no se ha utilizado bien la notación	Se ha planteado bien el modelo pero tiene demasiados errores	Se ha utilizado bien la notación, pero el diagrama no es del todo correcto o faltan clases y relaciones basándose en la especificación de requisitos	Se ha realizado correctamente el diagrama de clases y se ha utilizado bien la notación	20%	

	Insuficiente (0)	Mal (2)	Debe mejorar (4)	Cumple las expectativas (7)	Excelente (10)	Peso	Nota
Glosario de clases	No se ha realizado el glosario de clases		Se ha realizado, pero no se han descrito todas las clases o no se ha explicado su significado	Se han descrito todas las clases y se ha explicado su significado, pero no se incluye la descripción de sus principales atributos y métodos	Se han descrito todas las clases y se ha explicado correctamente su significado, atributos y métodos principales	4%	
Vista de interacción	No se han realizado los diagramas de secuencia	No se ha utilizado bien la notación o los diagramas tienen errores muy graves	Se han realizado menos de 10 diagramas de secuencia o no se ha utilizado bien la notación	Se han realizado entre 10-15 diagramas de secuencia y todos son correctos	Se han realizado más de 15 diagramas de secuencia y todos son correctos	20%	
Propuesta de arquitectura	No se ha realizado la propuesta arquitectónica		Se ha realizado, pero es incorrecta	Se ha realizado, pero no es óptima	Se ha realizado y es óptima	12%	
Glosario	No se ha realizado el glosario			Se ha realizado, pero tiene faltas de ortografía o no está expresado correctamente	Se ha realizado y está correctamente redactado	3%	
						TOTAL	

Si la práctica se supera con una nota mínima de un 5 quedará superada para futuros cursos académicos en caso de suspender la asignatura.

Las competencias que se asocian a esta actividad son las siguientes: CC1; CC2; CC8; CC16; IS2; IS4; TI1; CT2; CT3; CT4; CT5; CT8; CT9; CT10; CT11; CT12; CT13; CT14; CT16; CT17; CT18; CT19; CT20; CT21; CT22.

7.6.7. Modalidades para cursar la asignatura

Como se ha venido explicando, la docencia de la asignatura se ha organizado bajo un enfoque activo que obliga a los estudiantes a asistir y participar en las sesiones de clase que se han organizado tomando el trabajo final como hilo conductor.

Sin embargo, por más que este enfoque activo es más beneficioso para la formación de los estudiantes, es una aproximación que no todo el mundo está en condiciones de cursar, existen repetidores que pueden tener ya superada la práctica obligatoria, estudiantes que compaginan trabajo con sus estudios y no asisten a clase, incompatibilidades de horarios con asignaturas de otros cursos, estudiantes que se descuelgan de la opción de evaluación continua, etc.

Para facilitar que todos los estudiantes matriculados tengan la oportunidad de superar la asignatura se han definido dos itinerarios o modalidades. Estas modalidades se explican en la primera sesión de presentación de la asignatura y se les habilita un espacio en el campus virtual para que elijan cómo van a cursar la asignatura (apartado *Antes de comenzar*, ver Figura 7.38).

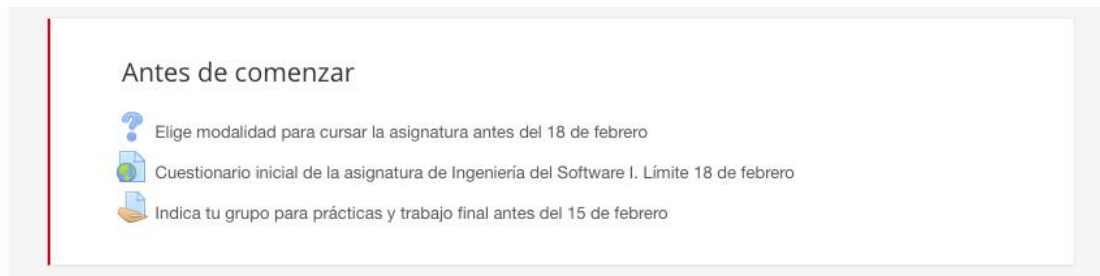


Figura 7.38. Captura del campus virtual del bloque en el que los estudiantes eligen la modalidad en la que van a cursar la asignatura

La modalidad o itinerario A es el que refleja un enfoque convencional orientado a una evaluación final. Tiene las siguientes características [117] (p. 16):

- No se realiza evaluación continua, es decir, este apartado de la calificación final o se tiene guardado de otros años o se pierde.
- La asistencia a las sesiones de clase es voluntaria y tiene un carácter mayormente pasivo, aunque se les invita a participar en los debates, preguntas, etc.
- Se pueden realizar los exámenes de tipo test parciales porque estos se orientan a eliminar materia del examen final, no computan en la evaluación continua.
- Se deberá realizar una entrega única de la práctica final y pueden ser requeridos para una defensa grupal de la misma.
- Está recomendada para aquellos que tengan conflicto de horario y, especialmente orientada a quienes tienen parte de la asignatura superada de cursos anteriores (práctica obligatoria y evaluación continua).

La modalidad o itinerario B es el que tiene un enfoque activo orientado a la evaluación continua. Tiene las siguientes características [117] (p. 17):

- La asistencia es obligatoria, al menos al 75% de las sesiones de teoría y práctica
- Se debe estar integrado en un grupo para el desarrollo de las actividades colaborativas.
- Se pueden realizar los exámenes de tipo test parciales.
- Se pueden realizar los ejercicios de modelado individuales, que computan para la evaluación continua.
- Se puede participar en los talleres de prácticas, que computan para la evaluación continua.
- La práctica obligatoria se realiza de forma incremental con tres entregas obligatorias (dos parciales y una final).
- Defensa del trabajo final bajo demanda del equipo docente.

7.6.8. Evaluación

La asignatura presenta una evaluación que combina la evaluación de las competencias que tienen que haber adquirido los estudiantes, llevando a la práctica los conocimientos adquiridos, con la demostración de una comprensión de los conocimientos fundamentales que se han impartido en la asignatura.

Tabla 7.9. Componentes de la evaluación final de la asignatura e hitos evaluables

Hito evaluable	Componente en la evaluación final						
	Conocimientos fundamentales (CF)			Práctica Final obligatoria (PF)			Evaluación continua (EC)
	Precondición	Nota	Comentario	Precondición	Nota	Comentario	
Examen conocimientos básicos: Test Parcial 1 (TP ₁)		TP ₁					
Examen conocimientos básicos: Test Parcial 2 (TP ₂)	SI TP ₁ ≥3	TP ₂	SI TP ₂ ≥3 Y ((TP ₁ +TP ₂)/2)≥4 ENTONCES CB=((TP ₁ +TP ₂)/2) SINO !CB				
Examen final (EF)	SI (CB≥4) Y (MPR≥4)	EF=(CB+MPR)/2	SI (CB<4) !CB para segunda convocatoria SI (MPR<4) !MPR para segunda convocatoria				
Examen Final Parte Conocimientos Básicos (Test) (CB)	SI !CB	CB					
Examen Final Parte Modelado y Preguntas Razonadas (MPR)	SI !MPR	MPR					
Trabajo Final (TF)				SI TF≥5	TF	SI TF≥5, TF para cualquier curso	
1 entrega				Modalidad A	TF=E ₁		
3 entregas				Modalidad B	TF=(E ₁ +E ₂ +E ₃)/3		
Ejercicios							$E_{jerc} = \frac{\sum_{i=1}^n e_i}{n}$
Defensa Taller (DT)							DT
Participación Taller (PT)							PT
Participación clase (PC)							PC
Presentación Público del Trabajo Final (PPTF)							PPTF
NOTA		CF = EF			PF = TF		EC = Ejerc+DT+PT+PC+PPTF
NOTA FINAL (NF)	NF = CF*0,4 + PF*0,35 + EC*0,25						
	SI NF≥5 Se ha superado la asignatura						

Se sigue en un enfoque que todas las actividades realizadas tengan un peso en la calificación final, computando en alguno de los tres apartados que intervienen en la misma: conocimientos fundamentales, práctica final obligatoria y evaluación continua. Además, se pide que se obtengan unos mínimos en la parte de conocimientos

fundamentales y en la práctica final obligatoria, lo que hace que la fórmula para el cálculo de la calificación final sea un tanto compleja, como se aprecia en la Tabla 7.9.

Para clarificar el procedimiento recogido en la Tabla 7.9 se va a proceder a explicar cada uno de los componentes que intervienen en el cálculo de calificación final.

El apartado de los conocimientos fundamentales se orienta a evaluar si el estudiante ha adquirido, por un lado, los conocimientos teóricos básicos de la asignatura, para lo que se va a utilizar el examen tipo test, y si los comprende para lo que se utilizarán preguntas cortas de respuesta abierta; por otro lado, en este apartado, el estudiante debe demostrar individualmente sus competencias a la hora de realizar modelos conceptuales, para lo que se le plantearán ejercicios de modelado.

El hito evaluable de los conocimientos fundamentales es el examen final de la asignatura, que tiene un peso en la calificación final de la misma de 0,4, pero para que esta parte compute en la calificación global debe haberse obtenido un mínimo de un 4 el examen. Este examen final, a su vez, se compone de dos partes que computan con el mismo peso en la nota del mismo, una parte de preguntas tipo test y una parte de problemas de modelado conceptual y de preguntas de respuesta abierta y corta. Para que la nota de este examen sea calculada se debe alcanzar una puntuación mínima de 4 puntos en cada una de las partes del mismo. Si en la primera convocatoria se supera el 4 en alguna de las partes, pero no en las dos, el estudiante puede decidir guardar esa nota parcial para la segunda convocatoria. Si después de las dos convocatorias no ha alcanzado un mínimo de un 4 en ambas partes del examen, esta parte no se habría superado y la asignatura estaría suspensa, no guardándose ninguna de las partes de este examen para futuros cursos.

Como la cantidad de materia y conceptos es elevada, se realizan dos pruebas parciales de tipo test, de forma que, si el estudiante saca una media superior o igual a 4 entre ambas, sin que ninguna de las notas sea inferior a un 3, habría eliminado la parte de preguntas tipo test del examen final y, por tanto, solo debería realizar la segunda parte del mismo, esto es, la parte problemas de modelado conceptual y de preguntas de respuesta abierta y corta.

La práctica final obligatoria tiene un peso en la calificación global de 0,35, siempre que se obtenga un mínimo de un 5. Tiene el cometido de evaluar las competencias prácticas de la asignatura y, fundamentalmente, la competencia transversa de trabajo en equipo.

Es la nota que se haya conseguido en la evaluación del trabajo final, que dependiendo de la modalidad puede constar de una sola entrega (modalidad A) o de tres entregas (modalidad B o de evaluación continua). Al igual que en el examen final, el estudiante tiene dos convocatorias para superar la práctica final obligatoria. Si se obtiene una nota superior o igual a 5 en la práctica obligatoria y el estudiante no supera la asignatura, puede guardar esta nota para cualquier curso futuro en el que se vuelva a matricular de la asignatura.

Con el apartado de evaluación continua se busca evaluar y valorar la participación activa del estudiante en las sesiones de clase presencial, tanto de teoría como de práctica, por lo que se exige un mínimo de asistencia a dichas sesiones del 75%. Esta parte no requiere un mínimo, se puede guardar para futuros cursos si no se supera la asignatura, pero no se puede recuperar dentro de un mismo curso académico porque es el resultado de la participación en clase. Se tienen varios hitos evaluables, el que más peso tiene es la evaluación de ejercicios cortos de modelado conceptual que se plantean y se recogen en clases aleatorias, con un mínimo de dos veces. En este apartado de evaluación continua computan las defensas de los talleres, la participación activa en las sesiones de teoría y de talleres, así como una presentación breve en público del trabajo final.

Los elementos de evaluación, congruentemente con lo que se ha presentado en la Tabla 7.9, se dividen en dos grupos. En primer lugar, se han definido 21 ítems de evaluación para todos los estudiantes. En segundo lugar, un conjunto de 4 ítems para los estudiantes que no aprobaron la asignatura en la primera convocatoria y que cuentan con una segunda convocatoria para mejorar sus calificaciones y aprobar la asignatura. La Tabla 7.10 muestra cada elemento con el tipo de valor (numérico para elementos cuantitativos y texto para elementos cualitativos), el rango de valores y una descripción.

Hay algunos elementos cualitativos que aparecen como cuantitativos en la Tabla 7.10 porque se han analizado utilizando rúbricas que transforman las medidas cualitativas en cuantitativas.

Tabla 7.10. Elementos usados para la evaluación de la asignatura. Fuente: Basada en: [271]

Ítem de evaluación	Tipo	Rango	Descripción
Ausencia	Numérico	0 – 47	Número de faltas a las clases presenciales de teoría y práctica
Participación	Texto	-	Comentarios sobre la participación en las clases presenciales
Taller 1	Numérico	0 – 1,25	Nota obtenida por presentar su solución o participar durante el primer taller
Taller 2	Numérico	0 – 1,25	Nota obtenida por presentar su solución o participar durante el segundo taller
Taller 3	Numérico	0 – 1,25	Nota obtenida por presentar su solución o participar durante el tercer taller
Informe del Taller	Numérico	0 – 1	Nota por entregar el informe del taller presentado previamente
Ejercicio 1	Numérico	0 – 10	Nota obtenida en el primer ejercicio UML (diagrama de clase) recogido como parte de la evaluación continua
Ejercicio 2	Numérico	0 – 10	Nota obtenida en el segundo ejercicio UML (diagrama de clase) recogido como parte de la evaluación continua
Evaluación continua	Numérico	0 – 10	La suma de: media del ejercicio 1 y ejercicio 2; taller 1, taller 2 y taller 3 con un máximo de 1.25 puntos; informe del taller; y presentación
Hito 1	Numérico	0 – 10	Resultado de la rúbrica para evaluar el hito 1 del trabajo final
Hito 2	Numérico	0 – 10	Resultado de la rúbrica para evaluar el hito 2 del trabajo final
Hito 3	Numérico	0 – 10	Resultado de la rúbrica para evaluar el hito 3 del trabajo final
Trabajo final	Numérico	0 – 10	Media de la evaluación de los tres hitos (o nota de la evaluación única para la modalidad de no evaluación continua)
Presentation	Numérico	0 – 1	Evaluación de la presentación en clase del trabajo final
Test 1	Numérico	0 – 10	Nota del primer test parcial sobre conceptos básicos
Test 2	Numérico	0 – 10	Nota del segundo test parcial sobre conceptos básicos
Nota global de los test parciales	Numérico	0 – 10	Media de los dos tests parciales
Examen final (parte test) C1	Numérico	0 – 10	Nota de la parte de test del examen final (primera convocatoria)
Examen final (parte modelado) C1	Numérico	0 – 10	Nota de la parte de modelado y preguntas de respuesta corta del examen final (primera convocatoria)
Nota global del examen final C1	Numérico	0 – 10	Media de las dos partes del examen final (primera convocatoria)
Calificación final de la asignatura C1	Numérico	0 – 10	Suma del 25% de la evaluación continua, 35% del proyecto final y 40% del examen final total y una parte subjetiva relacionada con la participación en clase, interés, etc. (primera convocatoria)
Examen final (parte test) C2	Numérico	0 – 10	Nota de la parte de test del examen final (segunda convocatoria)
Examen final (parte modelado) C2	Numérico	0 – 10	Nota de la parte de modelado y preguntas de respuesta corta del examen final (segunda convocatoria)
Nota global del examen final C2	Numérico	0 – 10	Media de las dos partes del examen final (segunda convocatoria)
Calificación final de la asignatura C2	Numérico	0 – 10	Suma del 25% de la evaluación continua, 35% del proyecto final y 40% del examen final total y una parte subjetiva relacionada con la participación en clase, interés, etc. (segunda convocatoria)

7.6.9. Tutorías

La acción tutorial es continua a través del campus virtual y bajo demanda si necesitan tratar aspectos que requieren consultar varios diagramas o el número de dudas es mayor, porque es más fácil y rápido encontrar un hueco común en las agendas de los profesores y de los estudiantes que tener que ceñirse a unas horas de consulta estipuladas que pueden no ser efectivas con respecto a los temas a abordar.

No obstante, el número de peticiones de tutoría presencial se reduce bastante por la interacción que se tiene en las sesiones de trabajo grupal en la práctica final, lo que les permite solventar muchas de sus dudas relativas a esta práctica en las horas presenciales de clase.

7.6.10. Recursos

Para el desarrollo de la asignatura se cuenta con una serie de recursos.

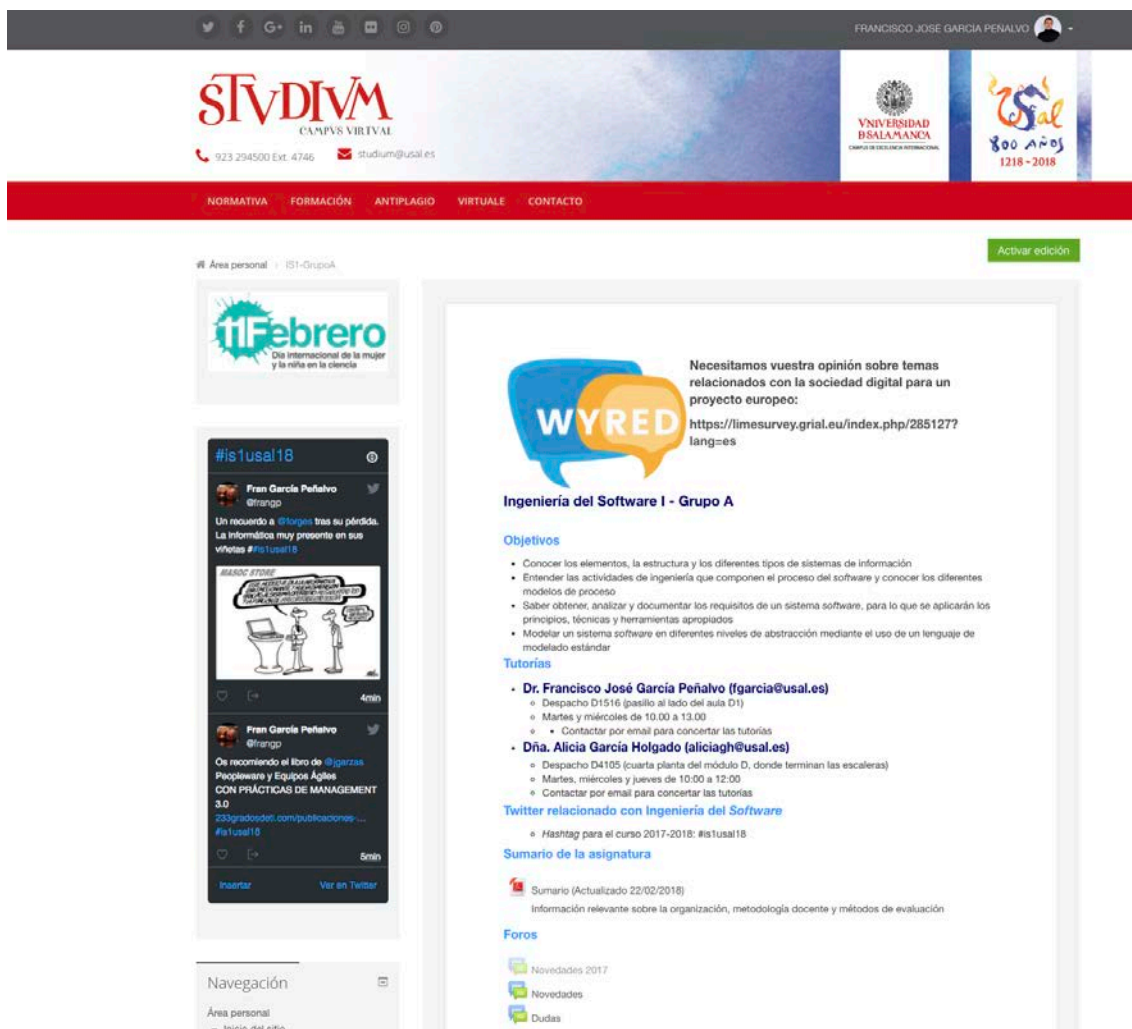


Figura 7.39. Captura con la parte inicial del espacio de la asignatura Ingeniería de Software I en el campus virtual Studium, donde destacan la información básica y los principales canales de interacción

El espacio de la asignatura en el campus virtual institucional, Studium, es el principal repositorio de materiales, a la vez que canal de información y medio de interacción. Ya se ha ido viendo en los subapartados anteriores como el campus virtual está muy presente en la organización de la docencia de esta asignatura, por más que sea una docencia presencial. En la Figura 7.39 se presenta la vista inicial de la asignatura en el campus virtual y en ella se pueden apreciar, además de la información básica y los foros principales, los tuits asociados al *hashtag* oficial de la asignatura (#is1usal18 para la edición del curso 2017-2018) o avisos especiales como la posible participación en el proyecto WYRED [272-274] aportando su opinión sobre la Sociedad Digital.

En la parte de contenidos están los materiales docentes de la asignatura, desarrollados por el profesorado de la asignatura y que se convierten en una de las principales fuentes de referencia [275]. Estos contenidos están disponibles y organizados en el campus virtual, pero además se cumple con el compromiso de los docentes con el conocimiento abierto [276], por lo que los materiales docentes están licenciados bajo licencia *Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional* (ver Figura 7.40). Para su libre distribución se ha creado una colección en el repositorio del Grupo de Investigación GRIAL (<https://goo.gl/M8JfNd>) y una comunidad en Zenodo (<https://goo.gl/XXS5t6>).

Licencia seleccionada

Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional



Figura 7.40. Licencia elegida para compartir los materiales docentes producidos por los docentes de la asignatura

En el desarrollo de los temas se recomiendan los capítulos u obras completas con las que completar el estudio de cada uno de ellos. Estas obras de referencia constituirían la bibliografía recomendada de la asignatura y que se muestra a continuación.

- G. Booch, J. Rumbaugh y I. Jacobson, 2ª, Ed. *El lenguaje unificado de modelado* (Object Technology Series). Madrid, España: Pearson Educación, 2007 [229].

- I. Jacobson, G. Booch y J. Rumbaugh, *El Proceso Unificado de desarrollo de software* (Object Technology Series). Madrid, España: Pearson Educación, 2000 [240].
- C. Larman, *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al Proceso Unificado*, 2ª ed. Madrid, España: Pearson Educación, 2003 [205].
 - De este libro existe una edición más moderna, pero no disponible en español:
 - C. Larman, *Applying UML and patterns. An introduction to object-oriented analysis and design and the Unified Process*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004 [206].
- S. L. Pfleeger, *Ingeniería del Software. Teoría y Práctica*. Argentina: Prentice Hall, 2002 [207].
- M. G. Piattini Velthius, J. A. Calvo-Manzano, J. Cervera Bravo y L. Fernández Sanz, *Análisis y Diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software*. Madrid, España: Ra-ma, 2004 [6].
- R. S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 7ª ed. México D. F., México: McGraw-Hill, 2010 [208].
 - De este libro existe una edición más moderna, pero no disponible en español:
 - R. S. Pressman y B. R. Maxim, *Software Engineering: A practitioner's approach*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2015 [5].
- J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy y W. Lorensen, *Modelado y diseño orientados a objetos. Metodología OMT*. Hertfordshire, UK: Prentice-Hall International, 1996 [247].
 - De este libro existe una edición más moderna, pero no disponible en español:
 - M. R. Blaha y J. R. Rumbaugh, *Object-oriented modeling and design with UML*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 2004 [107].

- J. Rumbaugh, I. Jacobson y G. Booch, *El Lenguaje Unificado de Modelado manual de referencia*, 2ª ed. (Object Technology Series). Madrid, España: Pearson Educación, 2007 [231].
- S. Sánchez Alonso, M. Á. Sicilia Urbán y D. Rodríguez García, *Ingeniería del Software. Un enfoque desde la guía SWEBOK*. Madrid, España: Ibergarceta Publicaciones, 2011 [190].
- I. Sommerville, *Ingeniería de Software*, 9ª ed. Naucalpan de Juárez, Estado de México, México: Pearson Educación, México, 2011 [2].
 - De este libro existe una edición más moderna, pero no disponible en español:
 - I. Sommerville, *Software Engineering*, 10th ed. Essex, England: Pearson Education Limited, 2016 [32].

Además, se incluye un conjunto reducido de referencias secundarias que pueden completar algún aspecto más concreto del temario o ampliar en foco hacia la asignatura de Ingeniería de Software II, concretamente:

- E. Gamma, R. Helm, R. Johnson y J. Vlissides, *Patrones de Diseño. Elementos de software orientado a objetos reutilizable*. Madrid, España: Pearson Educación, 2003 [277].
- J. Garzás, *Peopleware y equipos ágiles con prácticas de management 3.0*. Madrid, España: 233 grados de TI, 2017 [278].
- B. Meyer, *Construcción de software orientado a objetos*, 2ª ed. Madrid, España: Prentice Hall, 1999 [279].
- S. R. Schach, *Ingeniería de Software clásica y orientada a objetos*, 6ª ed. México: McGraw-Hill, 2006 [280].
- E. Yourdon, *Análisis Estructurado Moderno*. México: Prentice-Hall Hispanoamericana, 1993 [218].

También hay otras fuentes, diferentes a los libros, que son especialmente útiles en algunos temas. Se recomiendan las siguientes:

- A. Durán y B. Bernárdez, “Metodología para el análisis de requisitos de sistemas software (versión 2.2),” Universidad de Sevilla, Universidad de Sevilla, España, 2001. Disponible en: <https://goo.gl/youGje1> [281].

- A. Durán y B. Bernárdez, “Metodología para la elicitación de requisitos de sistemas software (versión 2.3),” Universidad de Sevilla, Universidad de Sevilla, España, Informe Técnico LSI-2000-10, 2002. Disponible en: <https://goo.gl/rhV8eV> [230].
- Object Management Group, “Unified Modeling Language specification version 2.5.1,” Object Management Group, Needham, MA, USA, formal/17-12-05, 2017. Disponible en: <https://goo.gl/kaE82a> [132].
- K. Pohl, “Requirements Engineering: An Overview,” en *Encyclopedia of Computer Science and Technology*, vol. 36, A. Kent y J. Williams, Eds., New York, USA: Marcel Dekker, 1997. Disponible en: <https://goo.gl/ojZsbN> [282].

Por último, aunque no se utiliza ninguna herramienta CASE en concreto y se deja total libertad para que cada estudiante decida con cuál o cuáles va a trabajar, se le ofrecen algunas recomendaciones:

- ArgoUML (<https://goo.gl/JnmivP>).
- Draw.io (<https://goo.gl/vRyrss>).
- Enterprise Architect (<https://goo.gl/uQ8yg6>).
- GenMyModel (<https://goo.gl/jXvQBK>).
- Modelio (<https://goo.gl/H9wPm5>).
- Microsoft Visio (<https://goo.gl/zqS9kc>).
- Rational DOORS (<https://goo.gl/Be2gVF>).
- REM (<https://goo.gl/Dtk59e>).
- Software Engineering Tutor – SET (<https://goo.gl/kWBaWM>) [283].
- Visual Paradigm (<https://goo.gl/Jin8Ew>).

7.6.11. Matriz de trazabilidad de las competencias

En la Tabla 7.11 se ha realizado una matriz de trazabilidad entre las competencias propuestas en la asignatura y los elementos de contenido y de actividad propuestos en la misma.

Tabla 7.11. Matriz de trazabilidad de las competencias propias de la asignatura Ingeniería de Software I

Competencia	Elemento de conocimiento / Actividad								Talleres	Práctica Final
	T1	T2	T3	T4	T5	T6	T7	T8		
CB5	•	•						•		
CC1			•	•					•	•
CC2			•	•	•	•	•			•
CC8				•			•			•
CC16	•		•		•	•				•
IS2				•					•	•
IS4				•			•		•	•
TI1	•	•			•	•				•
CT1	•	•	•	•	•	•	•	•	•	
CT2	•	•	•	•	•	•	•	•	•	•
CT3				•			•		•	•
CT4					•	•				•
CT5									•	•
CT8										•
CT9							•		•	•
CT10		•							•	•
CT11	•								•	•
CT12									•	•
CT13										•
CT14	•									•
CT16				•			•		•	•
CT17										•
CT18	•	•	•	•	•	•	•	•	•	•
CT19					•	•				•
CT20									•	•
CT21									•	•
CT22					•	•				•

7.7. Evaluación de la implementación del enfoque activo en la asignatura Ingeniería de Software I

7.7.1. Diseño de un instrumento de tipo pre-test / post-test

La evaluación del desempeño docente corre a cargo de la Unidad de Evaluación de la Calidad de la Universidad de Salamanca, que pasa encuestas a los estudiantes de los grados cada dos cursos académicos. Estos cuestionarios cumplen una función y dan cierta información al profesor sobre su desempeño docente, pero realmente la realimentación que este recibe sobre aspectos de grano fino es escasa o en muchos casos inexistente.

El cambio metodológico hacia un enfoque activo como el que se dio en la asignatura Ingeniería de Software I requiere de una información mucho más profunda que las encuestas oficiales. Los resultados obtenidos ya ofrecen un indicador obvio de referencia sobre el éxito o el fracaso de la innovación docente, pero tanto si funciona y se quiere mejorar como si no se obtienen los resultados esperados, se necesita un conocimiento mucho más detallado de cómo fue percibido y entendido el proceso de cambio.

Por este motivo se diseñaron un cuestionario pre-test y un cuestionario post-test [284] para evaluar el efecto del cambio metodológico llevado a cabo en la asignatura, con el objetivo de mejorar las calificaciones finales obtenidas por los estudiantes que cursan la asignatura. Se trata, pues, de evaluar el impacto conseguido al implementar una metodología activa en una asignatura que en cursos anteriores ha utilizado una metodología tradicional.

Los instrumentos se han elaborado a partir de una adaptación de trabajos previos y preguntas definidas *ad hoc*. En primer lugar, para el pre-test se ha utilizado el cuestionario *CEVEAPEU*, Cuestionario de Evaluación de las Estrategias de Aprendizaje de los Estudiantes Universitarios [285], elaborado y validado por investigadores de la Universidad de Valencia con el objetivo de proporcionar un instrumento más completo que los clásicamente utilizados para la evaluación de estrategias de aprendizaje. En segundo lugar, para medir el grado de satisfacción con las medidas implementadas, se ha utilizado un cuestionario de satisfacción publicado como anexo en la tesis doctoral “Evaluación del impacto de una metodología docente, basada en el aprendizaje activo del estudiante, en computación en ingenierías” realizada por la Dra. Dña. Ana Belén González Rogado [286].

El pre-test se compone de dos partes, un conjunto de preguntas de contexto definidas *ad hoc* y el cuestionario *CEVEAPEU*. El post-test se compone de tres partes: se mantiene una de las preguntas de contexto, relacionada con el grado de satisfacción con los estudios que se están realizando; el cuestionario *CEVEAPEU* para evaluar estilos de aprendizaje; y se incluye un conjunto de preguntas de satisfacción relacionadas con el cambio metodológico implementado.

La encuesta se puede utilizar en asignaturas relacionadas con la Ingeniería Informática con el fin de evaluar la implementación de una metodología activa en el plan de la asignatura. Aunque el segundo instrumento, el post-test, tiene preguntas centradas en las acciones llevadas a cabo en una asignatura en concreto, se puede modificar para adaptar el instrumento a otras asignaturas. En este caso se ha utilizado en el curso 2016-2017, concretamente en el grupo A de la asignatura Ingeniería de Software I, sin grupo de control.

Ambos cuestionarios (pre y post test)¹ se implementaron usando Google Forms. Tras aplicar los instrumentos, se consiguieron 51 respuestas válidas en el pre-test y 44 en el post-test.

A continuación, se van a mostrar algunos gráficos con los resultados de estos cuestionarios.

En la Figura 7.41 se recoge el sexo de las personas que contestaron el pre-test, 41 hombres (80,4%) y 10 mujeres (19,6%).

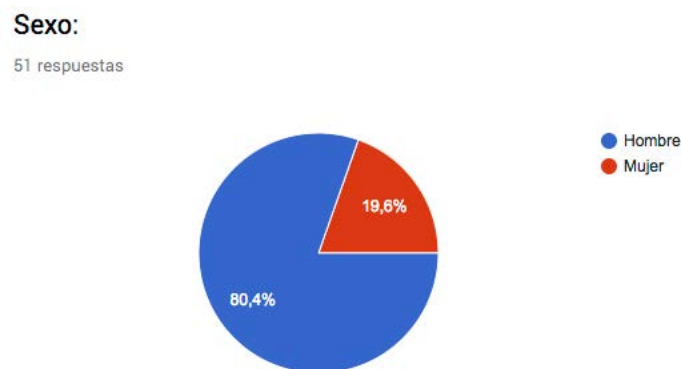


Figura 7.41. Sexo de los participantes en el pre-test

La Figura 7.42 presenta el curso más alto en el que están matriculados los estudiantes de la asignatura que contestaron el pre-test, en el que predomina el segundo curso con un 88,2% de las respuestas obtenidas.



Figura 7.42. Curso más alto en el que están matriculados de los participantes en el pre-test

¹ Para comparar los resultados de ambos instrumentos, se utilizó un identificador único que estaba presente tanto en el pre-test como en el post-test. Para ello, se pedía a los estudiantes que restaran de su DNI una fecha significativa para ellos y elegida por individualmente por cada uno de ellos (por ejemplo, su fecha de nacimiento).

La Figura 7.43 muestra la distribución de los años de nacimiento de los participantes en el pre-test.

Año de nacimiento

51 respuestas

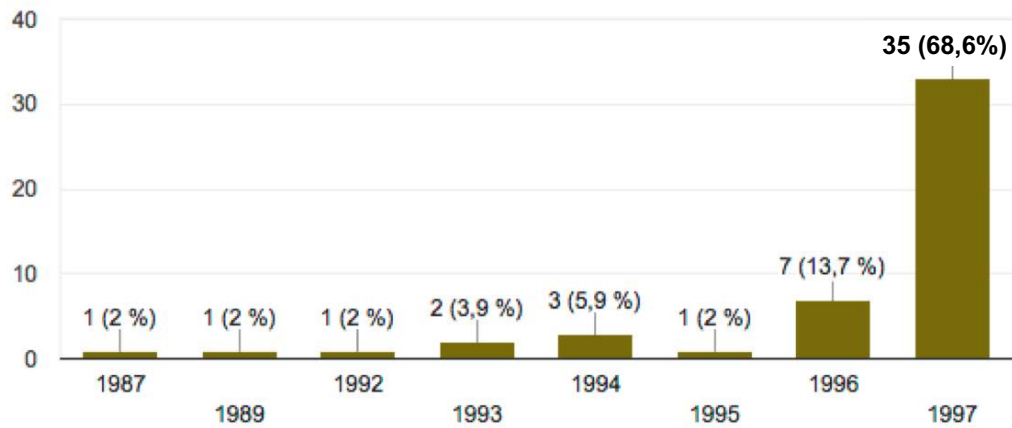


Figura 7.43. Año de nacimiento de los participantes en el pre-test

En la Figura 7.44 se muestra el orden de prioridad al elegir el Grado en Ingeniería Informática, el cual fue elegido como primera opción por el 90,2% de los participantes en el pre-test.

Elección de la carrera

51 respuestas

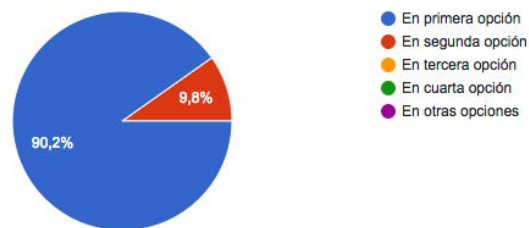


Figura 7.44. Orden de elección de la carrera de los participantes en el pre-test

En la Figura 7.45 se presenta el nivel de estudios de los padres de aquellos que contestaron el pre-test.

51 respuestas

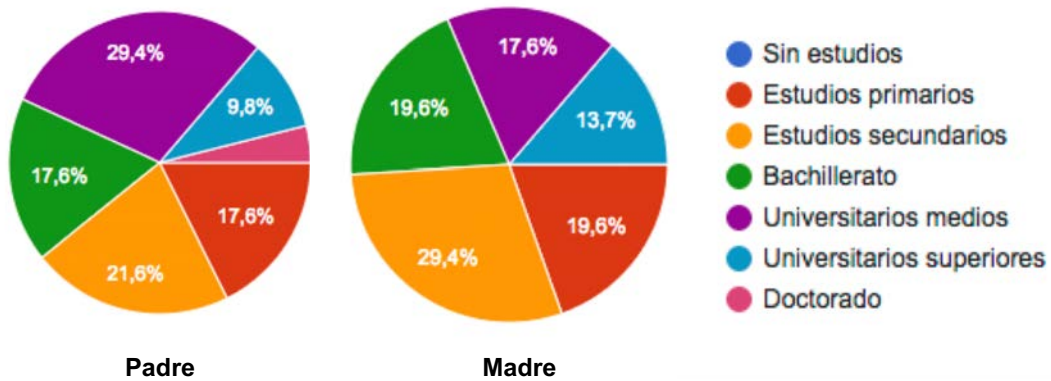


Figura 7.45. Nivel de estudios de los progenitores de los participantes en el pre-test

La Figura 7.46 recoge las notas de entrada en la Universidad, vía prueba de acceso, de los participantes en el pre-test.

Nota de entrada en la Universidad (PAU)

38 respuestas

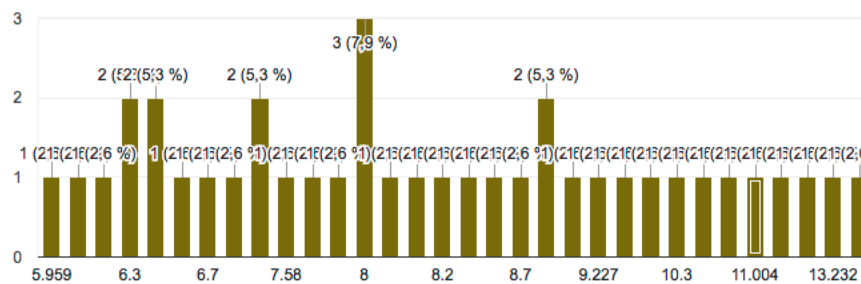
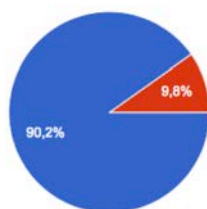


Figura 7.46. Nota de entrada en la Universidad (PAU) de los participantes en el pre-test

La Figura 7.47 presenta información sobre las veces que se han matriculado en la asignatura Ingeniería de Software I los estudiantes que han contestado el pre-test.

Es la primera vez que cursas la asignatura

51 respuestas



En caso negativo, ¿cuántas veces te has matriculado en esta asignatura?

7 respuestas

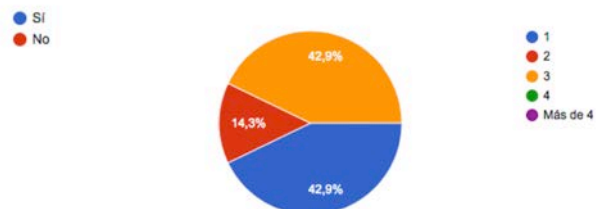


Figura 7.47. Información sobre el número de veces que se han matriculado en la asignatura los participantes en el pre-test

En la Figura 7.48 se recoge el grado de satisfacción con los estudios cursados en el Grado en Ingeniería Informática hasta el momento de comenzar la asignatura de Ingeniería de

Software I. En la Figura 7.50 se recoge la misma información de los participantes en el post-test, es decir, preguntada tras haber cursado esta asignatura.

Valora, en general, de 1 (nada satisfecho) a 5 (muy satisfecho), el grado de satisfacción, hasta el momento, con los siguientes aspectos de los estudios que estás realizando

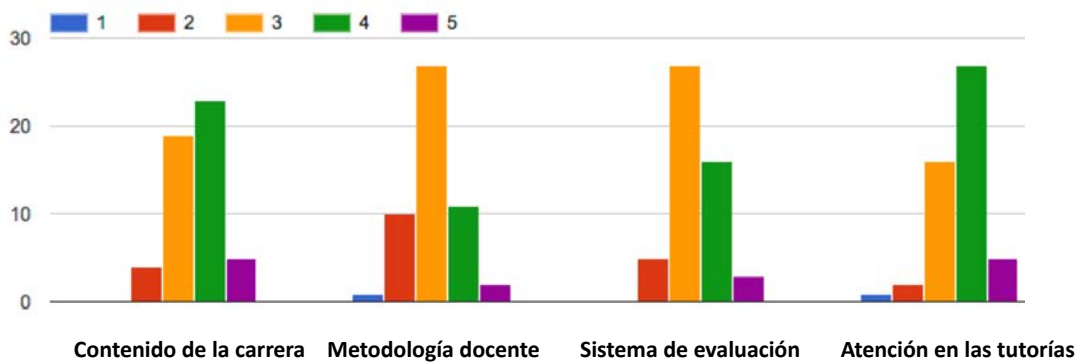


Figura 7.48. Satisfacción con los estudios realizados hasta el momento de comenzar la asignatura de Ingeniería de Software I de los participantes en el pre-test

En la Figura 7.49 se recogen algunas de las respuestas de los participantes en el pre-test a las preguntas relacionadas con sus estilos de aprendizaje. Lo propio con las respuestas de los participantes en el post-test se presenta en la Figura 7.51.

Lee atentamente las diversas cuestiones y selecciona la opción de respuesta que te resulte más próxima o que mejor se ajuste a tu situación. Ten en cuenta que no hay respuestas correctas ni incorrectas.

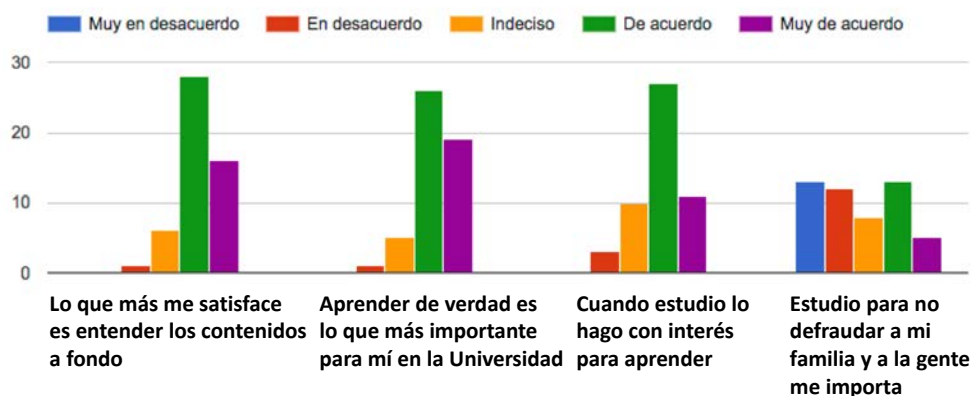


Figura 7.49. Preguntas relacionadas con los estilos de aprendizaje de los estudiantes participantes en el pre-test

1. Valora, en general, de 1 (nada satisfecho) a 5 (muy satisfecho), el grado de satisfacción, hasta el momento, con los siguientes aspectos de los estudios que estás realizando

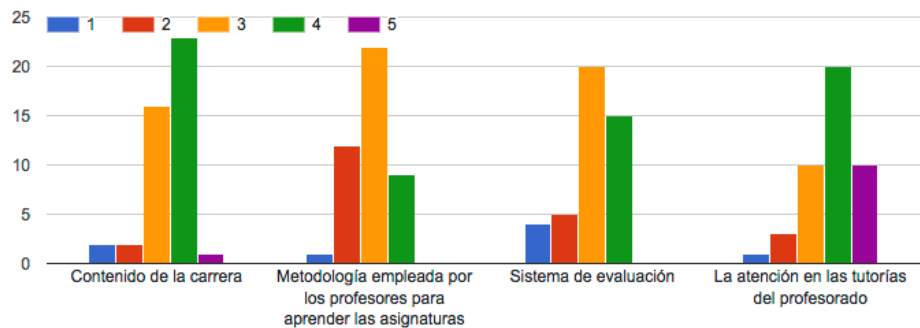


Figura 7.50. Satisfacción con los estudios realizados hasta el momento de haber finalizado la asignatura de Ingeniería de Software I de los participantes en el post-test

2. Lee atentamente las diversas cuestiones y selecciona la opción de respuesta que te resulte más próxima o que mejor se ajuste a tu situación

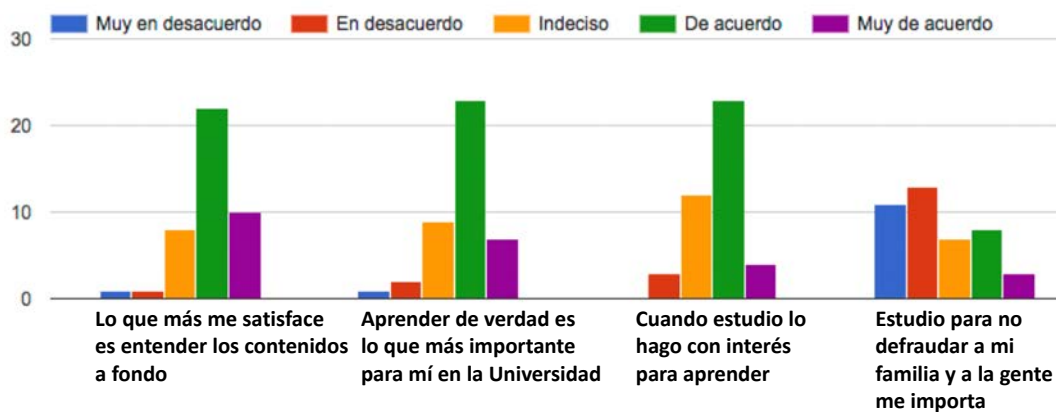


Figura 7.51. Preguntas relacionadas con los estilos de aprendizaje de los estudiantes participantes en el post-test

En la Figura 7.52 se recoge la opinión de los participantes en el post-test sobre su metodología de trabajo personal en el estudio de la asignatura Ingeniería de Software I, por su parte en la Figura 7.53 se ve su grado profundidad en el estudio de la asignatura.

3. Metodología de trabajo personal

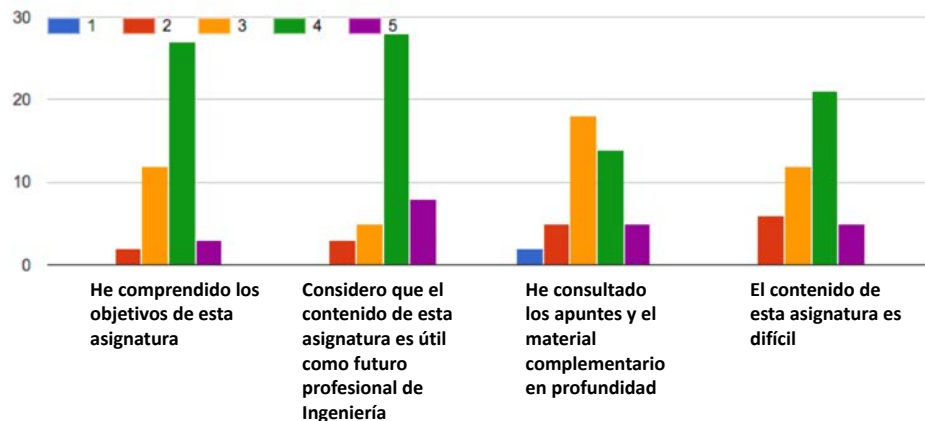


Figura 7.52. Valoración de la metodología personal de trabajo en la asignatura Ingeniería de Software I de los estudiantes participantes en el post-test

4. El grado de profundidad en el estudio del contenido ha sido el siguiente

39 respuestas

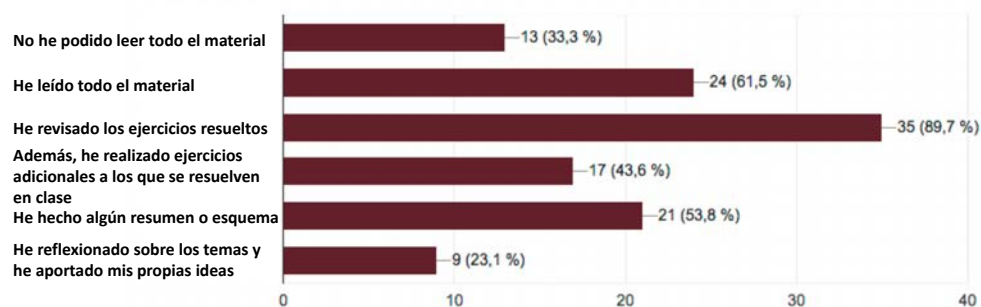


Figura 7.53. Grado de profundidad en el estudio de la asignatura Ingeniería de Software I de los estudiantes participantes en el post-test

En la Figura 7.54 se recoge la percepción que han tenido los estudiantes que han contestado el post-test sobre la metodología activa utilizada en la asignatura.

5. Percepción sobre la metodología activa



Figura 7.54. Percepción sobre la metodología activa seguida en asignatura Ingeniería de Software I por parte de los estudiantes participantes en el post-test

En la Figura 7.55 se presenta la satisfacción personal, en la Figura 7.56 la utilidad percibida sobre algunos de los recursos facilitados y en la Figura 7.57 la valoración de algunas de las actividades desarrolladas, siempre desde la perspectiva de aquellos que contestaron el post-test.

6. Satisfacción general

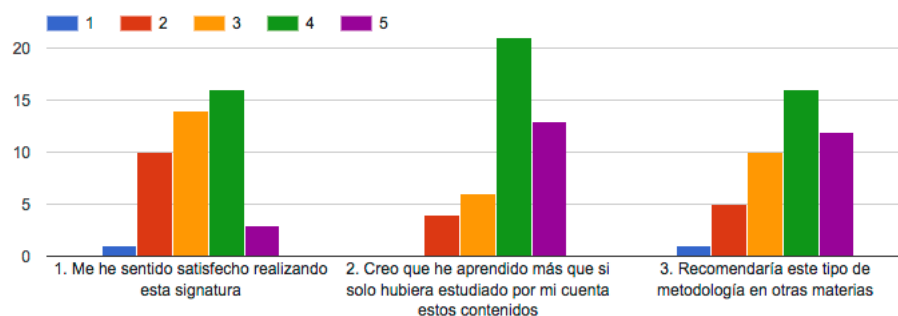


Figura 7.55. Satisfacción general con la asignatura Ingeniería de Software I por parte de los estudiantes participantes en el post-test

7. Utilidad para el estudio de la asignatura

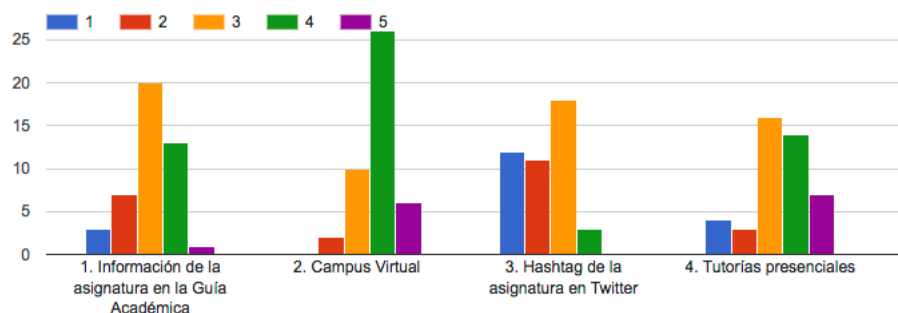


Figura 7.56. Utilidad percibida de algunos de los recursos para el estudio de la asignatura Ingeniería de Software I por parte de los estudiantes participantes en el post-test

8. Valoración dentro de la asignatura

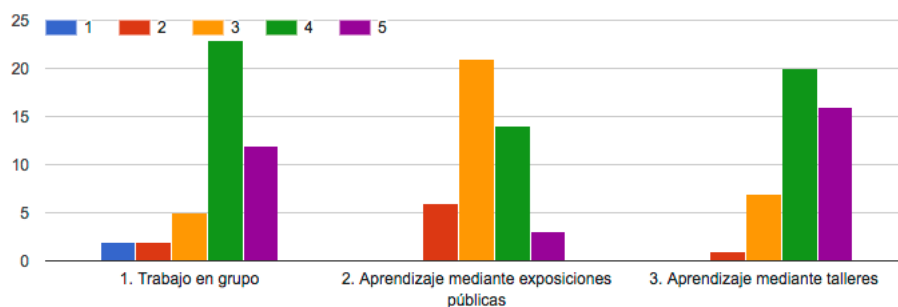


Figura 7.57. Valoración de algunas de las actividades realizadas en la asignatura Ingeniería de Software I por parte de los estudiantes participantes en el post-test

Las figuras siguientes dan una estimación del tiempo que invierten los estudiantes (desde la perspectiva de aquellos que han contestado el post-test) en las diferentes partes de la asignatura, concretamente en las clases presenciales (Figura 7.58), en las tutorías presenciales (Figura 7.59), en las tutorías virtuales (Figura 7.60), en horas de estudio (Figura 7.61), en preparar exámenes (Figura 7.62), en hacer ejercicios (Figura 7.63), en los talleres (Figura 7.64, Figura 7.65 y Figura 7.66) y en el trabajo final (Figura 7.67).

Horas presenciales clase (máx 18 h.)

44 respuestas

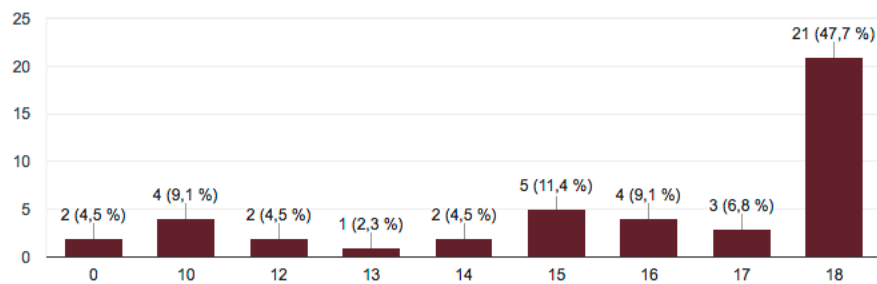


Figura 7.58. Tiempo invertido en ir a las clases presenciales (participantes en el post-test)

Tutorías presenciales

44 respuestas

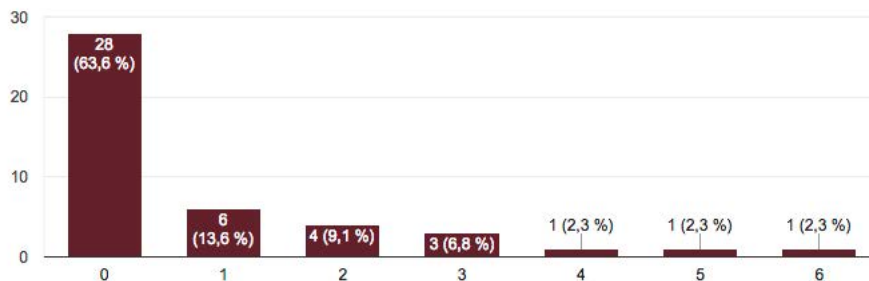


Figura 7.59. Tiempo invertido en las tutorías presenciales (participantes en el post-test)

Tutorías virtuales (email, foros)

44 respuestas

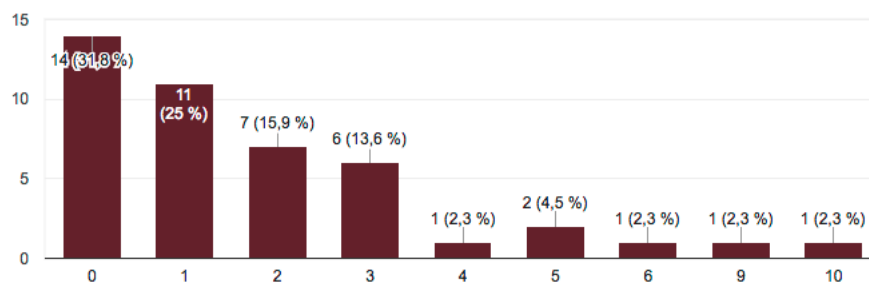


Figura 7.60. Tiempo invertido en las tutorías virtuales (participantes en el post-test)

Horas de estudio

44 respuestas

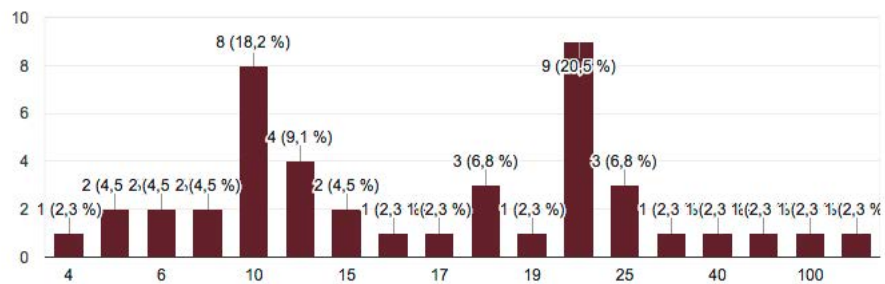


Figura 7.61. Tiempo invertido horas de estudio (participantes en el post-test)

Horas de examen

44 respuestas

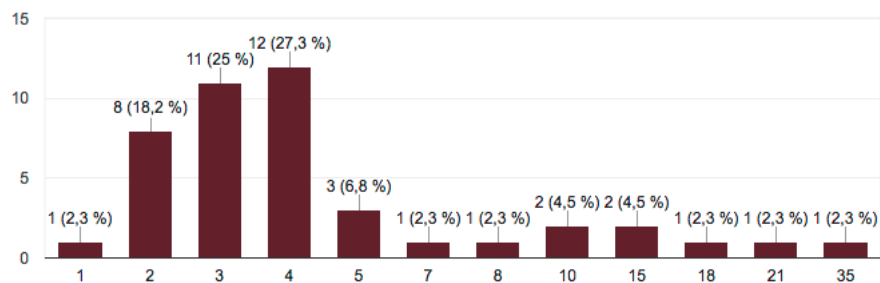


Figura 7.62. Tiempo invertido para preparar los exámenes (participantes en el post-test)

Tareas propuestas (ejercicios)

44 respuestas

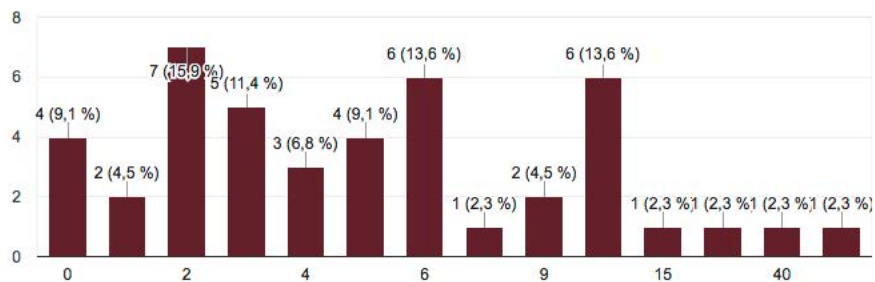


Figura 7.63. Tiempo invertido en hacer ejercicios (participantes en el post-test)

Taller 1 (Diagrama de casos de uso)

44 respuestas

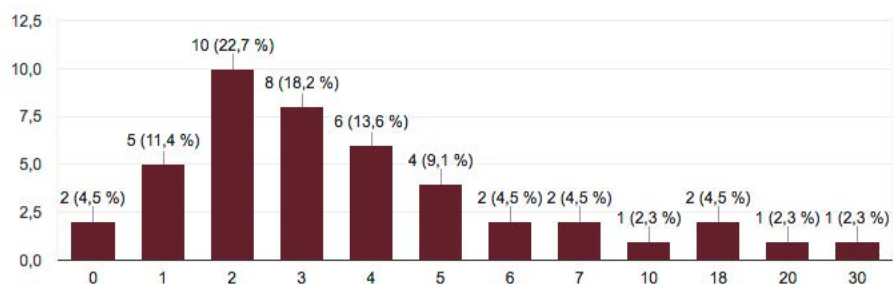


Figura 7.64. Tiempo invertido en el Taller 1 (participantes en el post-test)

Taller 2 (Diagrama de clases 1)

44 respuestas

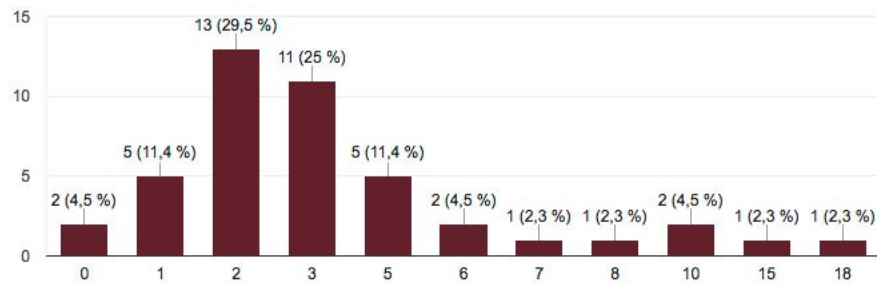


Figura 7.65. Tiempo invertido en el Taller 2 (participantes en el post-test)

Taller 3 (Diagrama de clases 2)

44 respuestas

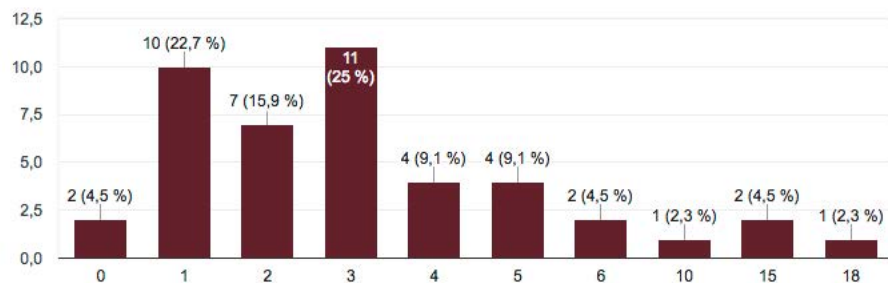


Figura 7.66. Tiempo invertido en el Taller 3 (participantes en el post-test)

Trabajo final

44 respuestas

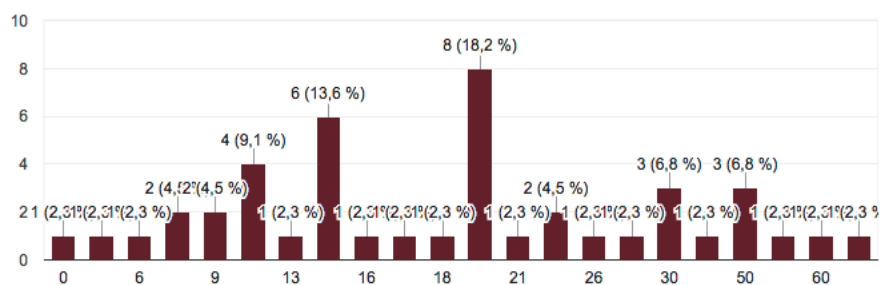


Figura 7.67. Tiempo invertido en el Trabajo Final (participantes en el post-test)

7.7.2. Análisis de los resultados obtenidos en el curso 2016-2017

El grupo A de la asignatura Ingeniería de Software I contó, en el curso 2016-2017, con 72 estudiantes, de los que 60 se matriculaba en la asignatura por primera vez (83,33%), cuatro lo hacían por segunda vez (5,55%), tres lo hacían por tercera vez (4,17%), tres se matriculaban por cuarta vez (4,17%) y dos se matriculaban por quinta vez (2,78%). Solo 10 eran mujeres (14%) y 62 hombres (86%). La mayoría de ellos tenían 20 años y todos estaban en el rango de edad de 20-30 años. 63 estudiantes (87,5%) seleccionaron la

modalidad de evaluación continua (Modalidad B), mientras que 9 (12,5%) seleccionaron la modalidad de evaluación final (Modalidad A).

El análisis de resultados se ha hecho poniendo el foco en la modalidad de evaluación continua (Modalidad B) [271], más concretamente en los hitos de aprendizaje de esta modalidad, como se presentó en la Tabla 7.10.

En la Tabla 7.12 se presentan los estadísticos descriptivos básicos por cada una de las pruebas realizadas a los estudiantes de Modalidad B (entre los que se incluyen el examen final, pero solo en la primera convocatoria). Las variables analizadas están comprendidas en un rango de 0-10 puntos, como suele ser habitual en el sistema de evaluación de estudiantes en España.

Tabla 7.12. Resultados descriptivos para estudiantes con Modalidad B (n=63). Fuente: Basada en: [271]

Variables	N	Mín.	Máx.	Md	Desv.Es.	Asim.	Curt.	Q1	Q2	Q3
Ejercicio 1	55	2,0	8,5	5,145	1,474	,109	,634	4,00	5,00	6,50
Ejercicio 2	62	,00	10,00	2,976	2,292	,937	,599	1,44	2,75	4,00
Test 1	63	,53	10,00	4,702	1,964	,112	,595	3,00	4,56	6,32
Test 2	50	1,17	9,33	4,330	1,692	,374	,662	3,17	4,33	5,21
Test Media	63	,26	9,66	4,069	1,976	,012	,595	2,86	4,30	5,47
E. Continua	63	,25	10,50	4,682	2,185	,516	,595	3,12	4,37	6,00
Trabajo Final	63	6,35	10,00	8,413	1,002	-,570	,595	7,50	8,70	9,20
Test C1	61	2,27	9,66	4,937	1,369	,451	,604	4,05	4,91	5,78
Supuestos C1	61	,50	8,75	4,746	2,152	-,219	,604	3,08	4,85	6,40

La nota más baja (como media) de las recogidas en la Tabla 7.12 se tiene en el ítem de evaluación “Ejercicio 2”, esto es debido a que el momento temporal en el que se recogió este ejercicio de modelado conceptual con diagramas de clase, que puntuaba en la evaluación continua, los estudiantes estaban trabajando en cerrar el hito 1, centrado en casos de uso. La nota más alta (como media) se tiene en el “Trabajo Final”, debido a que es el trabajo que hace de hilo conductor de la modalidad de evaluación continua, con realimentación continua sobre los hitos entregados que permiten a los estudiantes corregir sus errores y mejorar el trabajo de una manera continua. De igual forma, el ítem “Ejercicio 2” es el que muestra una mayor variabilidad en las notas, con una desviación estándar de 2,292. El motivo de esta dispersión es porque el modelado es una tarea que requiere una gran capacidad de abstracción que tarda en adquirirse y que debe sustentarse en unos profundos conocimientos del lenguaje de modelado, que si no están bien asentados por una práctica continua pueden causar problemas al llevarlos a la práctica. Por el contrario, el ítem de evaluación más homogéneo es el “Trabajo Final”, con una desviación estándar de 1,002, porque refleja la evaluación final de ese ítem entregado y corregido incrementalmente, con opciones de solventar los problemas

detectados en las entregas anteriores. Esta diferencia en variabilidad se observa también a través de las puntuaciones de los cuartiles 1, 2 y 3.

Como se puede observar en la Tabla de Normalidad (Tabla 7.13), todas las distribuciones correspondientes a las puntuaciones en pruebas de evaluación, a excepción del “Trabajo Final” ($p < 0,01$), siguen una distribución normal (n.s. $p < 0,05$), lo que permite aplicar tratamiento paramétrico en los siguientes análisis estadísticos multivariado.

Tabla 7.13. Resultados de la prueba de normalidad para la distribución de las variables del estudio. Fuente: Basada en: [271]

	Kolmogorov-Smirnov			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Ejercicio 1	,119	44	,134	,975	44	,432
Ejercicio 2	,121	44	,113	,949	44	,051
Test 1	,077	44	,200*	,971	44	,332
Test 2	,103	44	,200*	,975	44	,460
Test Media	,073	44	,200*	,958	44	,111
E. Continua	,110	44	,200*	,963	44	,163
Trabajo	,186	44	,001	,931	44	,011
Test C1	,060	44	,200*	,971	44	,321
Supuestos C1	,099	44	,200*	,958	44	,110

*. Este es un límite inferior de la significación verdadera
 a. Corrección de la significación de Lilliefors

En la Tabla 7.14 se comprueba la correlación entre las puntuaciones de las distintas pruebas (teniendo en cuenta que la auto-correlación no se considera, es decir, la variable “E. Continua” es resultado directo de otras dos variables “Ejercicio 1” y “Ejercicio 2” porque se deriva de ellas). Al analizar la tabla de correlaciones bivariadas, se observa que las correlaciones son significativas (n.s. $p < 0,05$) entre todas las variables (distintas de 0) y oscilan entre ,03 y 0,5 entre las variables no auto-correlacionadas. Esto da una idea de la relación que hay entre la producción de los estudiantes. Es decir, se observa una cierta coherencia en los resultados, quien es evaluado con calificación alta en los test, también lo hace en el resto de las pruebas y al revés.

Tabla 7.14. Correlaciones entre las puntuaciones de las distintas pruebas. Fuente: Basada en: [271]

		Ejercicio 1	Ejercicio 2	E. Continua	Test 1	Test 2	Test Media	Test C1	Supuestos C1
Ejercicio 1	Corr. de Pearson	1							
	N	55							
Ejercicio 2	Corr. de Pearson	,400**	1						
	N	54	62						
E. Continua	Corr. de Pearson	,626**	,778**	1					
	N	55	62	72					
Test 1	Corr. de Pearson	,443**	,404**	,454**	1				
	N	55	62	67	67				
Test 2	Corr. de Pearson	,270	,370**	,419**	,476**	1			
	N	45	49	52	52	52			
Test Media	Corr. de Pearson	,462**	,452**	,519**	,907**	,876**	1		
	N	55	62	72	67	52	72		
Test C1	Corr. de Pearson	,463**	,405**	,395**	,740**	,709**	,812**	1	
	N	55	60	68	65	52	68	68	
Supuestos C1	Corr. de Pearson	,432**	,430**	,508**	,580**	,468**	,603**	,621**	1
	N	55	60	68	65	52	68	68	68

** La correlación es significativa al nivel 0,01 (bilateral)

Finalmente, se va a analizar la variabilidad de la nota final de la primera convocatoria, Nota Final C1, que será la variable criterio, que se calcula a partir de las calificaciones en las diferentes pruebas, que serán las variables predictoras: Ejercicio 1, Ejercicio 2, E. Continua, Trabajo Final, Test 1, Test 2, Test Media, Test C1 y Supuestos C1. El objetivo es conocer cuál de todas las pruebas explica mayor porcentaje de la variabilidad de las notas finales en una primera convocatoria. Es decir, qué prueba es la que más discrimina entre los estudiantes que han pasado por la Modalidad B (aprendizaje activo) y más contribuye a la nota final.

Según se observa en la Tabla 7.15, después de realizar un análisis de regresión paso a paso (*stepwise regression*), el porcentaje de varianza del criterio explicada por los predictores en el modelo más exhaustivo es de un 89,7% y la variable que más explica es “Supuestos C1” (a).

Tabla 7.15. Resumen del modelo para la variable criterio: Nota Final C1. Fuente: Basada en: [271]

Modelo	R	R cuadrado	R cuadrado corregida	Error típico de la estimación	Estadísticos de cambio				
					Cambio en R cuadrado	Cambio en F	gl1	gl2	Sig. Cambio en F
1	,765 ^a	,585	,574	,95928	,585	50,805	1	36	,000
2	,869 ^b	,755	,741	,74807	,170	24,199	1	35	,000
3	,920 ^c	,847	,834	,59931	,092	20,531	1	34	,000
4	,939 ^d	,882	,868	,53388	,035	9,845	1	33	,004
5	,947 ^e	,897	,880	,50801	,014	4,447	1	32	,043

a. Variables predictoras: (Constante), Supuestos C1

b. Variables predictoras: (Constante), Supuestos C1, Trabajo Final

c. Variables predictoras: (Constante), Supuestos C1, Trabajo Final, Ejercicio 2

d. Variables predictoras: (Constante), Supuestos C1, Trabajo Final, Ejercicio 2, Test C1

e. Variables predictoras: (Constante), Supuestos C1, Trabajo Final, Ejercicio 2, Test C1, Ejercicio 1

El modelo de regresión se muestra en la Tabla 7.16. Se tiene en cuenta que el rango de puntuaciones es igual para todas las variables.

Tabla 7.16. Coeficientes (a) para el modelo final sobre la variable criterio Nota Final C1. Fuente: Basada en: [271]

Modelo	Coeficientes no estandarizados		Coeficientes tipificados	t	Sig.
	B	Error típico	Beta		
(Constante)	-1,249	,692		-1,805	,081
Supuestos C1	,236	,058	,330	4,059	,000
Trabajo Final	,430	,083	,321	5,153	,000
Ejercicio 2	,176	,039	,300	4,559	,000
Test C1	,271	,088	,246	3,075	,004
Ejercicio 1	,136	,065	,130	2,109	,043

a. Variable dependiente: Nota Final C1

De la Tabla 7.16 se deduce que la ecuación de regresión en puntuaciones directas es:

$$\hat{y} = -1,249 + 0,236 \text{ Supuestos C1} + 0,43 \text{ Trabajo Final} + 0,176 \text{ Ejercicio 2} + 0,271 \text{ Test C1} + 0,136 \text{ Ejercicio 1}$$

El mismo equipo docente coincidió en la impartición de la asignatura Ingeniería de Software I en el curso académico 2013-2014. En esa ocasión se aplicó una metodología tradicional. Los ítems de evaluación que utilizaron esta metodología fueron casi los

mismos ítems evaluados con la metodología activa en el curso académico 2016-1017 (Taller 1, Taller 2, Taller 3, Informe del Taller, Ejercicio 1, Ejercicio 2, Evaluación continua, Trabajo final, Test 1, Test 2, Test Media, Test C1, Supuestos C1, Nota Examen Final C1, Calificación Final C1, Test C2, Supuestos C2, Nota Examen Final C2, Calificación Final C2).

La comparación se realiza entre los resultados obtenidos por los estudiantes del Grupo A de la asignatura Ingeniería de Software I del curso académico 2013-2014 y los resultados obtenidos por los estudiantes que han cursado la Modalidad B durante el curso académico 2016-2017. En el curso 2013-2014, el Grupo A constaba de 79 estudiantes, de los que 50 se matricularon en la asignatura por primera vez (63,29%), 22 por segunda vez (27,85%) y 7 por tercera vez (8,86%). 17 estudiantes eran mujeres (21,52%) y 62 hombres (78,48%).

Una de las principales diferencias entre ambas experiencias es la puntuación obtenida en el trabajo final. En ambos cursos académicos los estudiantes se dividieron en grupos de tres miembros, excepto en casos particulares con dos miembros, para trabajar en el trabajo final. Aunque en 2013-2014 hubo más estudiantes, algunos de ellos se matricularon por segunda o más veces y decidieron mantener sus calificaciones relacionadas con el trabajo final. Finalmente, 23 grupos en 2013-2014 y 22 grupos en 2016-2017 tuvieron que entregar el trabajo final. En 2013-2014, 8 de los 23 trabajos finales tuvieron una calificación de suspenso o no se presentaron (34,78%); por el contrario, en 2016-2017 ningún trabajo final recibió una calificación de suspenso o no se presentó, es decir todos recibieron como poco la calificación de “Aprobado”. Además, en 2013-2014 no todos los trabajos finales se entregaron en primera convocatoria y solo se aprobaron 10 de los trabajos finales presentados en dicha, sin embargo, en la primera convocatoria de 2016-2017 se entregaron todos los trabajos finales y aprobaron 20 trabajos en primera convocatoria.

Tabla 7.17. Comparación de las notas del trabajo final en el curso 2013-2014 y en el curso 2016-2017.

Fuente: Basada en: [271]

Nota	2013-2014	2016-2017
Sobresaliente	1	6
Notable	6	14
Aprobado	8	2
Suspenso	5	0
No presentado	3	0
Total	23	22

La nota promedio de los trabajos finales que aprobaron en 2013-2014 fue de 6,4 sobre 10, mientras que en 2016-2017 fue de 8,37 sobre 10 (1,97 puntos de diferencia). Una comparación de las calificaciones en ambos cursos se muestra en la Tabla 7.17.

En cuanto al examen final en 2016-2017, 61 de los 63 estudiantes que cursaron la Modalidad B hicieron el examen final en primera convocatoria y 35 lo aprobaron (57,38%). Sin embargo, 7 de los 9 estudiantes que cursaron la Modalidad A (metodología tradicional) hicieron el examen y solo uno lo aprobó (14,29%). Esto supone un 50% de los estudiantes aprobaron la asignatura en primera convocatoria. Durante el curso 2013-2014, 65 de los 79 estudiantes hicieron el examen final en la primera convocatoria, pero solo 10 lo aprobaron (15,38% de los estudiantes que hicieron el examen final y 12,86% del número total de estudiantes matriculados); porcentaje similar al de los estudiantes en la modalidad A en 2016-2017.

Finalmente, en el curso 2016-2017 el 69,44% (50 de 72) de los estudiantes matriculados en la asignatura aprobaron, donde el 63,89% fueron estudiantes que seleccionaron la Modalidad B (46 de 72) y el 5,56% fueron estudiantes que seleccionaron la Modalidad A (4 de 72). Por otro lado, en 2013-2014 solo el 41,71% (33 de 79) aprobó la asignatura. La Figura 7.18 muestra una comparación entre las calificaciones finales en ambos cursos, con una metodología tradicional en 2013-2014 y una metodología activa en 2016-2017.

Tabla 7.18. Comparación de las notas finales entre el curso 2013-2014 y el curso 2016-2017. Fuente: Basada en: [271]

Nota	2013-2014	2016-2017
Matrícula de Honor	0	1
Sobresaliente	1	2
Notable	3	20
Aprobado	29	27
Suspense	34	12
No presentado	12	1
Total	79	63

Para determinar la igualdad o no en porcentajes de calificaciones, se ha realizado un contraste no paramétrico basado en la prueba de chi-cuadrado (nivel de significancia de $p < 0,05$). El contraste de la hipótesis ofrece un chi-cuadrado de 174,84, con 5 grados de libertad, con un valor asociado de probabilidad de aceptación de la hipótesis de igualdad menor que $p < 0,001$. Por tanto, existe una clara diferencia entre los resultados académicos de los dos cursos académicos, a favor de la experiencia realizada en 2016-2017, en la que se ha aplicado una metodología de aprendizaje activo. Más específicamente, la diferencia ocurre en la disminución del porcentaje de suspensos y no presentados y el porcentaje de notables, como se muestra en la Figura 7.68.

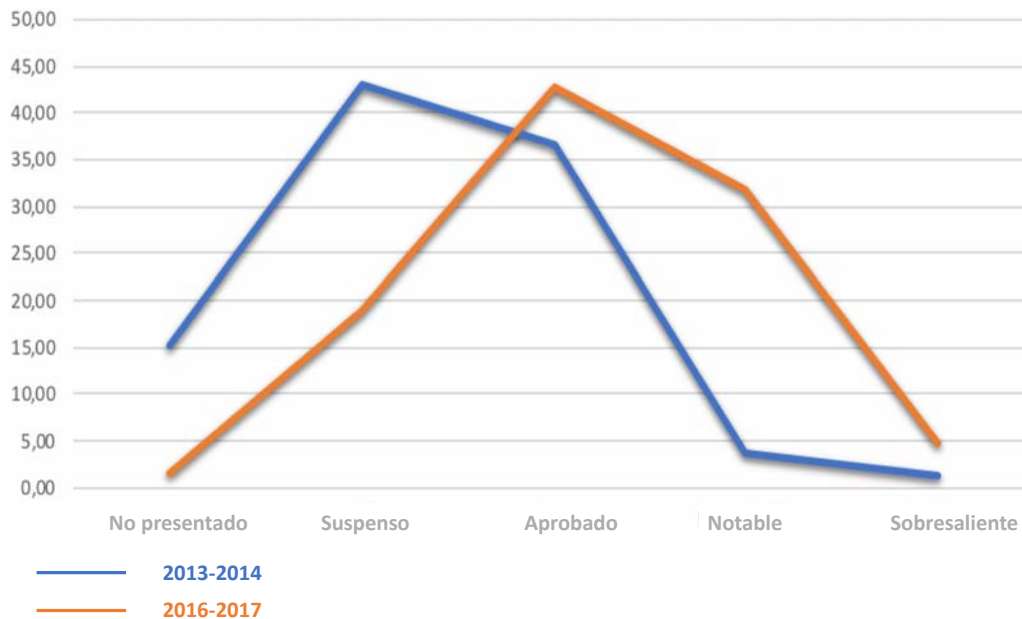


Figura 7.68. Comparación en términos porcentuales de la distribución del rendimiento académico entre los cursos 2013-2014 y 2016-2017

Tabla 7.19. Serie histórica de los resultados de los grupos impartidos de Ingeniería de Software I desde la implantación del Grado en Ingeniería Informática

Asignatura: Ingeniería del Software I											
Código: 101118											
Centro: Facultad de Ciencias											
Carácter: Troncal											
Grado en Ingeniería en Informática											
Convocatorias			Matriculados	Presentados	Tasa rendimiento	Tasa éxito	% sobre presentados				
							SU	AP	NT	SB	MH
Curso	2011-2012 Grupo B	1ª	42	31	0,21	0,29	(n=22) 70,97%	(n=7) 22,58%	(n=2) 6,45%	(n=0) 0%	(n=0) 0%
		2ª	33	24	0,36	0,50	(n=12) 50%	(n=12) 50%	(n=0) 0%	(n=0) 0%	(n=0) 0%
	2012-2013 Grupo B	1ª	75	64	0,2	0,23	(n=49) 75,56%	(n=12) 18,75%	(n=3) 4,69%	(n=0) 0%	(n=0) 0%
		2ª	60	47	0,32	0,40	(n=28) 59,57%	(n=15) 31,91%	(n=4) 8,51%	(n=0) 0%	(n=0) 0%
	2013-2014 Grupo A	1ª	83	66	0,17	0,21	(n=52) 78,79%	(n=12) 18,18%	(n=1) 1,5%	(n=1) 1,5%	(n=0) 0%
		2ª	69	52	0,28	0,37	(n=33) 63,46%	(n=18) 34,61%	(n=1) 1,92%	(n=0) 0%	(n=0) 0%
	2013-2014 Grupo B	1ª	92	78	0,15	0,18	(n=64) 82,05%	(n=9) 11,54%	(n=5) 6,41%	(n=0) 0%	(n=0) 0%
		2ª	78	63	0,25	0,32	(n=43) 68,25%	(n=17) 26,98%	(n=3) 4,76%	(n=0) 0%	(n=0) 0%
	2014-2015 Grupo B	1ª	98	83	0,33	0,39	(n=51) 61,45%	(n=17) 20,48%	(n=11) 13,25%	(n=3) 3,61%	(n=1) 1,20%
		2ª	66	46	0,32	0,46	(n=25) 54,35%	(n=15) 32,61%	(n=6) 13,04%	(n=0) 0%	(n=0) 0%
	2015-2016 Grupo B	1ª	97	83	0,16	0,19	(n=67) 80,72%	(n=12) 14,46%	(n=4) 4,82%	(n=0) 0%	(n=0) 0%
		2ª	81	67	0,38	0,46	(n=35) 52,24%	(n=22) 32,84%	(n=9) 13,43%	(n=0) 0%	(n=0) 0%
	2016-2017 Grupo A	1ª	76	68	0,49	0,57	(n=31) 45,6%	(n=17) 25%	(n=17) 25%	(n=2) 2,9%	(n=1) 1,5%
		2ª	39	30	0,44	0,57	(n=13) 43,3%	(n=13) 43,3%	(n=4) 13,4%	(n=0) 0%	(n=0) 0%

Para cerrar este apartado, en la Tabla 7.19 se recoge la serie histórica de los resultados de los grupos impartidos de Ingeniería de Software I desde que se implantó el Grado en Ingeniería Informática en el curso 2011-2012. Estos datos se obtienen de las estadísticas resumen de las actas oficiales de cada convocatoria y puede existir alguna discrepancia

con los datos manejados en los análisis realizados. Esto se debe a que los análisis se han hecho con los datos reales de los estudiantes que participaron en el grupo impartido y las actas recogen personas que abandonaron la asignatura sin cursarla y acuerdos con los profesores del otro grupo, por el que personas que oficialmente aparecen en el acta de un grupo, efectivamente lo cursan y se examinan en el otro por mejor compatibilidad de horarios, de forma que al final se traslada la calificación al grupo donde aparecen.

7.8. Reflexión final

La Ingeniería de Software se ha constituido como una disciplina con un cuerpo de conocimiento propio que sirve como base para constituir los estudios de Ingeniería en Informática en España, apostando por la dimensión ingenieril sobre el resto propuestas curriculares internacionales, orientadas a la ciencia de la computación, la ingeniería de computadores, los sistemas de información o las tecnologías de la información.

Como materia, la Ingeniería de Software dentro de los planes de estudios actuales relacionados con la Ingeniería en Informática de la Facultad de Ciencias de la Universidad de Salamanca cubre los aspectos fundamentales, los aspectos del modelado de requisitos y de diseño y los fundamentos de gestión de proyectos en el Grado; y aspectos avanzados de ingeniería conducida por modelos en el Máster Universitario en Ingeniería Informática o modelado de la ingeniería web en el Máster Universitario en Sistemas Inteligentes. Todas ellas han sido impartidas en algún momento por quien defiende la presente plaza de Catedrático de Universidad.

De todas estas asignaturas, se ha seleccionado Ingeniería de Software I porque es la primera asignatura del bloque de Ingeniería de Software y tiene el cometido de introducir los fundamentos de la materia y los fundamentos de la ingeniería de requisitos, desde su captura y gestión a su modelado y documentación. Además, se lleva impartiendo esta asignatura de introducción a la Ingeniería de Software ininterrumpidamente desde el curso 1996-1997 en la Universidad de Burgos y desde el curso 1998-1999 en la Universidad de Salamanca, es decir, 22 años.

La experiencia con esta asignatura hace que se conozca lo difícil que resulta esta materia para los estudiantes de Ingeniería en Informática. Las principales dificultades identificadas son:

- Es la primera asignatura que presenta a los estudiantes una dimensión de ingeniería de los estudios y que la relaciona con la profesión de ingeniero en informática.
- Es una asignatura que introduce unos métodos, técnicas y herramientas que son difíciles de asimilar con el tamaño de los trabajos que se pueden desarrollar en el ámbito de una asignatura y con el trabajo en grupo que, como mucho, involucra a dos o tres estudiantes.
- Los conceptos y la actividad de modelado requieren de una capacidad de abstracción que estudiantes de segundo de grado no han desarrollado completamente.
- El modelado y la documentación son tareas menos atractivas que la programación.
- El estudiante, normalmente, no es hasta después de haber acabado los estudios y tener unos años de experiencia cuando empieza a entender, con suerte de la labor desempeñada, la necesidad del enfoque de ingeniería en el desempeño de la profesión.

Consciente de estas dificultades, durante estos años impartiendo la asignatura se han desarrollado y experimentado diferentes innovaciones orientadas a mejorar la atracción de los estudiantes hacia la asignatura y sus resultados académicos. Así, entre otras diversas iniciativas, se han definido planes de calidad [287, 288] y protocolos de evaluación por pares [289, 290], se ha definido un protocolo de sesión de taller basado en el aprendizaje basado en problemas [249], se han desarrollado herramientas CASE que facilitasen la realización de modelos y de documentación [283, 291-293], se han explorado estrategias de adaptación de la asignatura al EEES [202, 203, 294], se ha apostado por digitalizar contenidos y actividades para potenciar el campus virtual de la asignatura y para compartirlos en abierto [295], se ha explorado la realización de prácticas externas virtuales en empresas internacionales [296-298], se ha analizado el desempeño de los estudiantes de la asignatura utilizando analíticas de aprendizaje [299-306] o se ha apostado por concienciar sobre la perspectiva del género en los estudios y en la profesión de ingeniero en informática [266-269].

Sin embargo, la innovación que más calado y mejores resultados ha venido con el cambio metodológico para implantar una metodología activa basada en el aprendizaje basado en problemas [271], que toma la práctica final obligatoria como el hilo conductor

de la asignatura, para lo que se planifican contenidos y las actividades de forma que alimenten el desarrollo de este trabajo en grupo. La planificación compagina una aproximación basada en las fases de Inicio y Elaboración del Proceso Unificado [62] con uso de técnicas propias de los métodos ágiles [278] y del Proceso *Software* Personal (PSP) [307, 308].

Ya se conocían múltiples ejemplos de la aplicación de métodos activos en diversos campos [251, 309-312] y especialmente en la ingeniería [250, 286, 313, 314] y en la asignatura de Ingeniería de Software I desde su primera edición se habían aplicado ciertas partes que buscaban la participación activa de los estudiantes, pero no fue hasta el curso 2016-2017 cuando se hizo el cambio completo hacia el enfoque activo.

El cambio metodológico ha permitido aumentar el número de estudiantes que aprueban la asignatura en la primera convocatoria. En particular, el porcentaje ha aumentado aproximadamente un 20%, del 41,71% al 63,89%, con una alta tasa de éxito en los estudiantes que seleccionaron la modalidad con un enfoque de aprendizaje activo. Además, el promedio de las calificaciones finales ha aumentado en 1,49 puntos sobre 10.

En cuanto al trabajo final, que ha de desarrollarse en grupos, el aprendizaje activo a través del aprendizaje basado en proyectos ha permitido alcanzar una tasa de éxito del 100% en la modalidad basada en aprendizaje activo, con una nota promedio de 8,37 sobre 10 puntos. El porcentaje de aprobados ha aumentado alrededor del 35% entre el curso 2013-2014 y el curso 2016-2017.

Además, el número de estudiantes que han superado el examen final en la primera convocatoria también ha aumentado sobre un 35%, del 12,86% al 50%. Además, el porcentaje de aprobados en el curso 2013-2014 es similar al porcentaje de aprobados en la modalidad de no evaluación continua en 2016-2017, ambos con una metodología tradicional.

Referencias

- [1] P. Naur y B. Randell Eds., "Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968." Brussels, Belgium: Scientific Affairs Division, NATO, 1969. Disponible en: <https://goo.gl/PYKmfY>.
- [2] I. Sommerville, *Ingeniería de Software*, 9ª ed. Naucalpan de Juárez, Estado de México, México: Pearson Educación, México, 2011.
- [3] IEEE, *IEEE Standard Glossary of Software Engineering Terminology* (ANSI/ IEEE Std 729-1983). USA: IEEE, 1983. doi: 10.1109/IEEESTD.1983.7435207.
- [4] IEEE, *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Std 610.12-1990). USA: IEEE, 1990. doi: 10.1109/IEEESTD.1990.101064.
- [5] R. S. Pressman y B. R. Maxim, *Software Engineering: A practitioner's approach*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2015.
- [6] M. G. Piattini Velthius, J. A. Calvo-Manzano, J. Cervera Bravo y L. Fernández Sanz, *Análisis y Diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software*. Madrid, España: Ra-ma, 2004.
- [7] M. G. Piattini Velthius, J. A. Calvo-Manzano, J. Cervera Bravo y L. Fernández Sanz, *Análisis y diseño detallado de aplicaciones informáticas de gestión*. Madrid, España: Ra-ma, 2007.
- [8] B. Mynatt, *Software engineering with students project guidance*. Upper Saddle River, NJ, USA: Prentice-Hall International, 1990.
- [9] B. Randell, "Memories of the NATO Software Engineering Conferences," *IEEE Annals of the History of Computing*, vol. 20, no. 1, pp. 51-54, 1998.
- [10] B. Randell y J. N. Buxton Eds., "Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969." Brussels, Belgium: Scientific Affairs Division, NATO, 1970.

- [11] J. E. Tomayko, "A Historian's view of software engineering," en *Proceedings of the Thirteenth Conference on Software Engineering Education and Training (Austin, TX, USA, 6-8 March 2000)* pp. 101-102, USA: IEEE, 2000. doi: 10.1109/CSEE.2000.827027.
- [12] C. Lewerentz y H. Rust, "Are software engineers true engineers?," *Annals of Software Engineering*, vol. 10, no. 1, pp. 311-328, 2000/11/01 2000. doi: 10.1023/A:1018952103240.
- [13] A. Bryant, "Metaphor, myth and mimicry: The bases of software engineering," *Annals of Software Engineering*, vol. 10, no. 1, pp. 273-292, 2000/11/01 2000. doi: 10.1023/A:1018947902331.
- [14] M. Shaw, "Prospects for an engineering discipline of software," *IEEE Software*, vol. 7, no. 6, pp. 15-24, 1990. doi: 10.1109/52.60586.
- [15] M. Shaw y J. E. Tomayko, "Models for Undergraduate Project Courses in Software Engineering," en *Software Engineering Education*, J. E. Tomayko, Ed. Lecture Notes in Computer Science, no. 536, pp. 33-71, Berlin, Heidelberg: Springer, 1991. doi: 10.1007/BFb0024284.
- [16] F. L. Bauer, "Software Engineering," en *Information Processing 71: Proceedings of IFIP Congress 71, Ljubljana, Yugoslavia, August 23-28, 1971*, C. V. Freiman, J. E. Griffith y J. L. Rosenfeld, Eds., Amsterdam, The Netherlands: North Holland, 1972.
- [17] J. M. Buxton, P. Naur y B. Randell Eds., "Software Engineering Concepts and Techniques (Proceedings of 1968 NATO Conference on Software Engineering)." New York, NY, USA: Van Nostrand Reinhold, 1976.
- [18] D. L. Parnas, "Software engineering or methods for the multi-person construction of multi-version programs," en *Programming Methodology. IBM 1974*, C. E. Hackl, Ed. Lecture Notes in Computer Science, no. LNCS 23, pp. 225-235, Berlin, Heidelberg: Springer-Verlag, 1975. doi: 10.1007/3-540-07131-8_28.
- [19] D. L. Parnas, "Software Engineering: Multi-person Development of Multi-version Programs," en *Dependable and Historic Computing: Essays Dedicated to Brian Randell on the Occasion of His 75th Birthday*, C. B. Jones y J. L. Lloyd, Eds. Lecture Notes in Computer Science, no. LNCS 6875, pp. 413-427, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-24541-1_31.
- [20] B. W. Boehm, "Software Engineering," *IEEE Transactions on Computers*, vol. C-25, no. 12, pp. 1226-1241, 1976. doi: 10.1109/TC.1976.1674590.
- [21] M. V. Zelkowitz, A. C. Shaw y J. D. Gannon, *Principles of Software Engineering and Design*. Englewoods Clif, NJ, USA: Prentice-Hall, 1979.
- [22] R. Fairley, *Software Engineering Concepts*. New York, NY, USA: McGraw-Hill, 1985.
- [23] Comité de Calidad del Software, "Glosario de Términos de Calidad e Ingeniería del Software," Asociación Española para la Calidad, 1987.
- [24] W. S. Humphrey, *Managing the software process*. Reading, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [25] ISO/IEC/IEEE, *Systems and software engineering - Vocabulary (ISO/IEC/IEEE 24765:2010(E))*. USA: IEEE, 2010. doi: 10.1109/IEEESTD.2010.5733835.
- [26] W. B. Frakes, C. Fox y B. A. Nejme, *Software Engineering in the UNIX/C Environment*. Englewood Cliffs, NJ, USA: Prentice Hall, 1991.
- [27] B. I. Blum, *Software engineering. A holistic view*. New York, NY, USA: Oxford University Press, 1992.
- [28] A. M. Davis, *Software requirements. Objects, functions and states*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall International, 1993.

- [29] W. S. Humphrey, "Software Engineering," en *Encyclopedia of Computer Science*, A. Ralston y E. D. Reilly, Eds. pp. 1217-1222, New York, NY, USA: Van Nostrand Reinhold, 1993.
- [30] M. Shaw y D. Garlan, *Software architecture: Perspectives on an emerging discipline*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1996.
- [31] B. Boehm, "A view of 20th and 21st century software engineering," en *Proceedings of the 28th international conference on Software engineering, ICSE '06 (Shanghai, China - May 20 - 28, 2006)* pp. 12-29, New York, NY, USA: ACM, 2006. doi: 10.1145/1134285.1134288.
- [32] I. Sommerville, *Software Engineering*, 10th ed. Essex, England: Pearson Education Limited, 2016.
- [33] The British Computer Society y The Institution of Electrical Engineering, *A report on undergraduate curricula for Software Engineering Curricula*. UK: The British Computer Society and The Institution for Electrical Engineers, 1989.
- [34] G. Ford, "1990 SEI Report on Undergraduate Software Engineering Education," Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA, USA, Technical Report, 1990. Disponible en: <https://goo.gl/z6BN0K>.
- [35] C. A. R. Hoare, "Software engineering," *BCS, Computer Bulletin*, vol. 2, no. 6, pp. 6-7, 1975.
- [36] J. A. McDermid Ed. "The software engineer's reference book." Oxford, UK: Butterworth Heinemann, 1991.
- [37] R. S. Pressman, *Software Engineering: A Practitioner's Approach - European Adaptation*, 5th ed. London, England: McGraw-Hill, 2000.
- [38] I. Sommerville, *Software Engineering*, 6th ed. Boston, MA, USA: Addison-Wesley, 2001.
- [39] A. I. Wasserman, "Toward a discipline of software engineering," *IEEE Software*, vol. 13, no. 6, pp. 23-31, 1996. doi: 10.1109/52.542291.
- [40] L. V. Bertalanffy, *General Systems Theory: Foundations, development, applications*. New York, NY, USA: George Brazillier, 1968.
- [41] J. L. Le Moigne, *Les Systèmes d'information dans les organisations*. Paris, France: Presses universitaires de France, 1973.
- [42] B. Meyer, *Object Oriented Software Construction*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1997.
- [43] B. W. Boehm, *Software Engineering Economics*. Upper Saddle River, NJ, USA: Prentice Hall, 1981.
- [44] B. W. Boehm, "Software Engineering Economics," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 1, pp. 4-21, 1984. doi: 10.1109/TSE.1984.5010193.
- [45] ISO/IEC/IEEE, *ISO/IEC/IEEE International Standard - Systems and software engineering -- Software life cycle processes*, 2nd ed. (ISO/IEC 12207:2008(E), IEEE Std 12207-2008). USA: IEEE, 2008. doi: 10.1109/IEEESTD.2008.4475826.
- [46] ISO/IEC/IEEE, *ISO/IEC/IEEE International Standard - Systems and software engineering -- Software life cycle processes (ISO/IEC/IEEE 12207:2017(E))*. USA: IEEE, 2017. doi: 10.1109/IEEESTD.2017.8100771.
- [47] ISO/IEC/IEEE, *ISO/IEC/IEEE International Standard - Systems and software engineering -- System life cycle processes (ISO/IEC/IEEE 15288:2015(E))*. USA: IEEE, 2015. doi: 10.1109/IEEESTD.2015.7106435.
- [48] IEEE Computer Society, *IEEE Guide--Adoption of ISO/IEC TR 24474:2010 Systems and Software Engineering-- Life Cycle Management--Guidelines for Process Description (IEEE Std 24774-2012)*. New York, NY, USA: IEEE, 2012. doi: 10.1109/IEEESTD.2012.6190704.

- [49] J. Nogueira, C. Jones y Luqi, "Surfing the Edge of Chaos: Applications to Software Engineering," presentado en Command and Control Research and Technology Symposium, Naval Post Graduate School, Monterey, CA, USA, 2000. Disponible: <https://goo.gl/vXoAjK>.
- [50] S. Kauffman, *At Home in the Universe. The Search for the Laws of Self-Organization and Complexity*. New York, NY, USA: Oxford University Press, 1995.
- [51] D. Oliver y J. Roos, "The poised organization: Navigating effectively in knowledge landscapes," presentado en Strategic Management Society Annual Conference, Phoenix, AR., USA, 1996.
- [52] C. Perrow, "A Framework for Comparative Analysis of Organizations," *American Sociological Review*, vol. 32, pp. 194-208, 1967.
- [53] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," presentado en Western Electronic Show and Convention (WesCon) August 25–28, 1970, Los Angeles, CA, USA, 1970.
- [54] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," en *Proceedings of the 9th international conference on Software Engineering, ICSE'87 (Monterey, California, USA)* pp. 328-338, Los Alamitos, CA, USA: IEEE Computer Society Press, 1987.
- [55] W. W. Agresti Ed. "New paradigms for software development." Los Alamitos, CA, USA: IEEE Computer Society Press, 1986.
- [56] W. W. Agresti, "The conventional software life cycle model: Its evolution and assumptions," en *New paradigms for software development*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1986.
- [57] R. Balzer, T. J. Cheatham y C. Green, "Software Technology in the 1990's: Using a New Paradigm," *Computer*, vol. 16, no. 11, pp. 39-45, 1983. doi: 10.1109/MC.1983.1654237.
- [58] D. R. Graham, "Incremental development: review of nonmonolithic life-cycle development models," *Information and Software Technology*, vol. 31, no. 1, pp. 7-20, 1989/01/01/ 1989. doi: 10.1016/0950-5849(89)90049-9.
- [59] B. Boehm, "A spiral model of software development and enhancement," *SIGSOFT Software Engineering Notes*, vol. 11, no. 4, pp. 14-24, 1986. doi: 10.1145/12944.12948.
- [60] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61-72, 1988. doi: 10.1109/2.59.
- [61] J. C. Derniame, B. A. Kaba y D. Wastell Eds., "Software process: Principles, methodology, and technology," Lecture Notes in Computer Science LNCS 1500. Berlin, Heidelberg: Springer-Verlag, 1999. doi: 10.1007/3-540-49205-4.
- [62] I. Jacobson, G. Booch y J. Rumbaugh, *The Unified Software Development Process* (Object Technology Series). Reading, Massachusetts, USA: Addison Wesley, 1999.
- [63] P. Kruchten, *The Rational Unified Process. An Introduction*, 3rd ed. (Object Technology Series). Boston, MA, USA: Addison-Wesley, 2004.
- [64] K. Beck *et al.*, "Agile Manifesto," Agile Alliance, 2001, Disponible en: <https://goo.gl/uXEUwJ>.
- [65] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70-77, 1999. doi: 10.1109/2.796139.
- [66] K. Beck, *Extreme Programming Explained. Embrace Change*. Boston, MA, USA: Addison-Wesley, 2000.
- [67] K. Schwaber y M. Beedle, *Agile software development with Scrum*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

- [68] K. Schwaber, *Agile Project Management with Scrum* (Developer Best Practices). Redmond, WA, USA: Microsoft Press, 2004.
- [69] J. Stapleton, *DSDM: Business Focused Development*, 2nd ed. (Agile Software Development Series). London, UK: Pearson Education, 2003.
- [70] Agile Alliance. (2018). *Agile Practices Timeline*. Disponible en: <https://goo.gl/gTxQbb>.
- [71] Real Academia Española, *Diccionario de la lengua española. Versión electrónica 23.1*, Madrid, España: Real Academia Española, 2017. [Online]. Disponible en: <https://goo.gl/Hkxjid>.
- [72] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy y W. Lorensen, *Object-oriented modeling and design*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.
- [73] B. Henderson-Sellers y D. G. Firesmith, "Comparing OPEN and UML: The two third-generation OO development approaches," *Information and Software Technology*, vol. 41, no. 3, pp. 139-156, 1999. doi: 10.1016/S0950-5849(98)00127-X.
- [74] M. A. Jackson, *Principles of program design*. Orlando, FL, USA: Academic Press, 1975.
- [75] M. A. Jackson, *System development*. London, UK: Prentice-Hall International, 1983.
- [76] J. R. Cameron, *JSP and JSD: The Jackson Approach to Software Development*, 2nd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1989.
- [77] J. Warnier, *Logical Construction of Programs*. New York, NY, USA: Van Nostrand Reinhold, 1974.
- [78] K. T. Orr, *Structured System Development*. New York, NY, USA: Yourdon Press, 1977.
- [79] K. T. Orr, *Structured Requirements Definition*. Topeka, KS, USA: Ken Orr & Associates, 1981.
- [80] E. Yourdon y L. Constantine, *Structured design*. New York, NY, USA: Yourdon Press, 1975.
- [81] E. Yourdon y L. L. Constantine, *Structured design: Fundamentals of a discipline of computer program and systems design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979.
- [82] E. Yourdon, *Modern structured analysis*. Englewood Cliffs, NJ, USA: Prentice Hall, 1989.
- [83] T. DeMarco, *Structured analysis and system specification* (Yourdon Press Computing Series). Upper Saddle River, NJ, USA: Prentice-Hall, 1979.
- [84] C. P. Gane y T. Sarson, *Structured systems analysis: Tools and techniques*. Englewood Cliffs, NJ, USA: Prentice Hall Professional Technical Reference, 1979.
- [85] C. P. Gane y T. Sarson, *Structured systems analysis and design*. New York, NY, USA: Improved Systems Technologies, 1977.
- [86] H. Tardieu, A. Rochfeld y R. Coletti, *La Méthode Merise. Tome 1. Principes et outils*. Paris: Editions d'Organisation, 1983.
- [87] H. Tardieu, A. Rochfeld, R. Coletti, G. Panet y G. Vahée, *La Méthode Merise. Tome 2. Démarche et pratiques*. Paris: Editions d'Organisation, 1985.
- [88] A. Rochfeld y J. Morejon, *La Méthode Merise. Tome 3. Gamme opératoire*. Paris: Editions d'Organisation, 1989.
- [89] Yourdon Inc., *Yourdon™ Systems Method. Model-driven systems development* (Yourdon press Computing Series). Englewood Cliffs, NJ, USA: Prentice Hall, 1993.
- [90] C. Ashworth y M. Goodland, *SSADM: A Practical Approach*. London, UK: McGraw-Hill, 1990.

- [91] Central Computer & Telecommunications Agency, *SSADM Foundation* (Business Systems Development with SSADM). London, UK: Stationery Office Books, 2000.
- [92] Ministerio de las Administraciones Públicas, *Metodología Métrica 2.1. Volúmenes 1-3*. Madrid, España: Tecnos, 1995.
- [93] Ministerio de las Administraciones Públicas, *Métrica v3*, Madrid, España: Ministerio de las Administraciones Públicas, 2001. [Online]. Disponible en: <https://goo.gl/FZ3aX4>.
- [94] P. T. Ward y S. J. Mellor, *Structured development for real-time systems: Vol. I: Introduction and tools* (Yourdon Press Computing Series). Upper Saddle River, NJ, USA: Prentice Hall, 1985.
- [95] P. T. Ward y S. J. Mellor, *Structured development for real-time systems: Vol. II: Essential modeling techniques* (Yourdon Press Computing Series). Upper Saddle River, NJ, USA: Prentice Hall, 1985.
- [96] S. J. Mellor y P. T. Ward, *Structured development for real-time systems: Vol. III: Implementation modeling techniques* (Yourdon Press Computing Series). Upper Saddle River, NJ, USA: Prentice Hall, 2000.
- [97] D. J. Hatley y I. A. Pirbhai, *Strategies for real-time system specification*. New York, NY, USA: Dorset House Publishing, 1987.
- [98] B. J. Wielinga, A. T. Schreiber y J. A. Breuker, "KADS: A modeling approach to knowledge engineering," *Knowledge Acquisition*, vol. 4, pp. 5-23, 1992.
- [99] A. T. Schreiber, B. J. Wielinga y J. A. Breuker Eds., "KADS: A principled approach to knowledge-based system development," *Knowledge-Based Systems Book Series*. London, UK: Academic Press, 1993.
- [100] A. T. Schreiber, B. J. Wielinga, J. M. Akkermans, W. Van De Velde y R. de Hoog, "CommonKADS. A comprehensive methodology for KBS development," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 9, no. 6, p. 28–37, 1994. doi: 10.1109/64.363263.
- [101] A. Gómez-Pérez, N. Juristo, C. Montes y J. Pazos, *Ingeniería del conocimiento. Diseño y Construcción de Sistemas Expertos*. Madrid, España: Ceura, 1997.
- [102] M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta y A. Gangemi Eds., "Ontology Engineering in a Networked World." Berlin Heidelberg: Springer-Verlag, 2012. doi: 10.1007/978-3-642-24794-1.
- [103] M. C. Suárez-Figueroa, A. Gómez-Pérez y M. Fernández-López, "The NeOn methodology for ontology engineering," en *Ontology Engineering in a Networked World*, M. C. Suárez-Figueroa, A. Gómez-Pérez, E. Motta y A. Gangemi, Eds. pp. 9-34, Berlin Heidelberg: Springer-Verlag, 2012. doi: 10.1007/978-3-642-24794-1_2.
- [104] G. Booch, *Object-oriented analysis and design*, 2nd ed. Redwood City, CA, USA: The Benjamin/Cummings Publishing Company, 1994.
- [105] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen y K. A. Houston, *Object-oriented analysis and design with applications*, 3rd ed. (Object-Technology). Boston, MA, USA, 2007.
- [106] J. Rumbaugh, *OMT insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*. New York, NY, USA: SIGS Books Publications, 1996.
- [107] M. R. Blaha y J. R. Rumbaugh, *Object-oriented modeling and design with UML*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 2004.
- [108] I. Jacobson, M. Christerson, P. Jonsson y G. Övergaard, *Object oriented software engineering: A use case driven approach*. Reading, MA, USA: Addison-Wesley, 1992.

- [109] D. Coleman, P. Arnold, S. Bodoff, C. Dolin, F. Hayes y P. Jeremaes, *Object-Oriented Development: The Fusion Method*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.
- [110] B. Henderson-Sellers y J. M. Edwards, *BOOKTWO of object-oriented knowledge: The working object: Object-Oriented Software Engineering: Methods and Management* (Prentice-Hall Object-Oriented Series). Englewood Cliff, NJ, USA: Prentice-Hall, 1994.
- [111] B. Henderson-Sellers y J. M. Edwards, "MOSES: A Second Generation Object-Oriented Methodology," *Object Magazine*, vol. 4, no. 2, pp. 68-73, 1994.
- [112] O. Pastor, E. Insfrán, V. Pelechano, J. Romero y J. Merseguer, "OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods," en *Advanced Information Systems Engineering (9th International Conference on Advanced Information Systems Engineering, CAiSE97 - Barcelona, Catalonia, Spain, June 1997)*, A. Olivé y J. A. Pastor, Eds. Lecture Notes in Computer Science, no. LNCS 1250, pp. 145-158, Berlin, Heidelberg: Springer, 1997. doi: 10.1007/3-540-63107-0_11.
- [113] O. Pastor, J. Gómez, E. Insfrán y V. Pelechano, "The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming," *Information Systems*, vol. 26, no. 7, pp. 507-534, 2001. doi: 10.1016/s0306-4379(01)00035-7.
- [114] O. Pastor y I. Ramos, *Oasis 2.1.1: A class-definition language to model information systems using an object-oriented approach*. Valencia, Spain: Servicio de Publicaciones UPV, Universidad Politécnica de Valencia, 1995.
- [115] P. Sánchez, P. Letelier, I. Ramos y O. Pastor, "OASIS 3.0: Un enfoque formal para el modelado conceptual orientado a objeto," en *Actas de las III Jornadas de Trabajo MENHIR (13 y 14 de noviembre de 1998, Murcia, España)*, B. Moros y J. Sáez, Eds. pp. 63-74, Murcia, España: Proyecto MENHIR, 1998.
- [116] P. Letelier, P. Sánchez, O. Pastor y I. Ramos, *OASIS Versión 3: Un enfoque formal para el modelado conceptual orientado a objeto*. Valencia, España: Servicio de Publicaciones UPV, Universidad Politécnica de Valencia, 1998.
- [117] F. J. García-Peñalvo y A. García-Holgado, "Sumario de la asignatura Ingeniería de Software I," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/aKvUGZ>. doi: 10.5281/zenodo.1183772.
- [118] J. H. Canós Cerdá, *Proyecto Docente e Investigador. Trabajo de Investigación. Catedrático de Universidad. Ingeniería del Software, Integración e Interoperabilidad. Área de Lenguajes y Sistemas Informáticos*. Valencia, España: Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 2017.
- [119] F. P. Brooks, *The mythical man-month. Essays on software engineering. Anniversary Edition*. Boston, MA, USA: Addison Wesley, 1995.
- [120] E. W. Dijkstra, "Letters to the editor: go to statement considered harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147-148, 1968. doi: 10.1145/362929.362947.
- [121] C. Böhm y G. Jacopini, "Flow diagrams, turing machines and languages with only two formation rules," *Communications of the ACM*, vol. 9, no. 5, pp. 366-371, 1966. doi: 10.1145/355592.365646.

- [122] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576-580, 1969. doi: 10.1145/363235.363259.
- [123] E. W. Dijkstra, "Co-operating sequential processes," en *Programming Languages*, F. Genuys, Ed. pp. 43-112, New York, NY, USA: Academic Press, 1968.
- [124] F. T. Baker, "Structured programming in a production programming environment," *ACM SIGPLAN Notices*, vol. 10, no. 6, pp. 172-185, 1975. doi: 10.1145/390016.808437.
- [125] U.S. Department of Defense. (1985). *MIL-STD-1521B: Reviews and audits*. Washington, DC, USA: U.S. Department of Defense.
- [126] U.S. Department of Defense. (1988). *DOD-STD-2167A: Defense system software development*. Washington, DC, USA: U.S. Department of Defense.
- [127] M. C. Paulk, "A history of the Capability Maturity Model for software," *Software Quality Professional*, vol. 12, no. 1, pp. 5-19, 2009.
- [128] M. A. Cusumano, "The software factory: a historical interpretation," *IEEE Software*, vol. 6, no. 2, pp. 23-30, 1989. doi: 10.1109/MS.1989.1430446.
- [129] F. J. García-Peñalvo, *Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos* (Vitor, no. 53). Salamanca, España: Ediciones Universidad de Salamanca, 2000.
- [130] G. Booch, J. Rumbaugh y I. Jacobson, 2nd, Ed. *The Unified Modeling Language User Guide* (Object Technology Series). Upper Saddle River, NJ, USA: Addison-Wesley, 2005.
- [131] J. Rumbaugh, I. Jacobson y G. Booch, *The Unified Modeling Language reference manual*, 2ª ed. (Object Technology Series). Boston, MA, USA: Addison Wesley, 2005.
- [132] Object Management Group, "Unified Modeling Language specification version 2.5.1," Object Management Group, Needham, MA, USA, formal/17-12-05, 2017. Disponible en: <https://goo.gl/kaE82a>.
- [133] F. P. J. Brooks, "No silver bullet. Essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10-19, 1987. doi: 10.1109/MC.1987.1663532.
- [134] T. J. Berners-Lee, R. Cailliau y J.-F. Groff, "The world-wide web," *Computer Networks and ISDN Systems*, vol. 25, no. 4-5, pp. 454-459, 1992. doi: 10.1016/0169-7552(92)90039-S.
- [135] E. Gamma, R. Helm, R. Johnson y J. Vlissides, *Design patterns: Elements of reusable object oriented software* (Addison-Wesley Professional Computing Series). Boston, MA, USA: Addison-Wesley, 1995.
- [136] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad y M. Stal, *Pattern-oriented software architecture. A system of patterns. Volume 1* (Wiley Series in Software Design Patterns). Chichester, West Sussex, England: John Wiley & Sons, 1996.
- [137] L. Bass, P. Clements y R. Kazman, *Software architecture in practice*, 3rd ed. (The SEI Series in Software Engineering). Upper Saddle River, NJ, USA: Addison-Wesley, 2013.
- [138] K. Lyytinen, L. Mathiassen y J. Ropponen, "A framework for software risk management," *Journal of Information Technology*, vol. 11, no. 4, pp. 275-285, 1996. doi: 10.1057/jit.1996.2.
- [139] R. N. Charette, *Software engineering risk analysis and management*. New York, NY, USA: McGraw-Hill, 1989.
- [140] B. W. Boehm y R. Ross, "Theory-W software project management principles and examples," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, pp. 902-916, 1989. doi: 10.1109/32.29489.

- [141] R. M. Stallman, *Free Software, Free Society - Selected Essays of Richard M. Stallman*, 2nd ed. Boston. USA: GNU Press, 2010. Disponible en: <https://goo.gl/u29sjx>.
- [142] L. Torvalds y A. Diamond, *Just for fun: The story of an accidental revolutionary*. New York, NY, USA: Harper, 2002.
- [143] G. Moody, *Rebel code: Linux and the open source revolution*. New York, NY, USA: Basic Books, 2002.
- [144] E. S. Raymond, *The Cathedral and the Bazaar*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999.
- [145] A. Capiluppi y M. Michlmayr, "From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects," en *Open Source Development, Adoption and Innovation. IFIP International Conference on Open Source Systems, OSS 2007*, J. Feller, B. Fitzgerald, W. Scacchi y A. Sillitti, Eds. IFIP - The International Federation for Information Processing, no. 234, pp. 31-44, Boston, MA: Springer doi: 10.1007/978-0-387-72486-7_3.
- [146] B. Shneiderman, *Software psychology: Human factors in computer and information systems* (Winthrop computer systems series). Cambridge, MA, USA: Winthrop Publishers, 1980.
- [147] L. Bass y J. Coutaz, *Developing software for the user interface* (SEI Series). Reading, MA, USA: Addison-Wesley, 1991.
- [148] M. C. Paulk, B. Curtis, M. B. Chrissis y C. V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software*, vol. 10, no. 4, pp. 18-27, 1993. doi: 10.1109/52.219617.
- [149] M. C. Paulk, C. V. Weber, B. Curtis y M. B. Chrissis, *The Capability Maturity Model: Guidelines for improving the software process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
- [150] J. Highsmith, *Adaptive software development. A collaborative approach to managing complex systems*. New York, NY, USA: Dorset House, 2000.
- [151] X. Larrucea, R. V. O. Connor, R. Colomo-Palacios y C. Y. Laporte, "Software Process Improvement in Very Small Organizations," *IEEE Software*, vol. 33, no. 2, pp. 85-89, 2016. doi: 10.1109/MS.2016.42.
- [152] S. Biffel, A. Aurum, B. W. Boehm, H. Erdogmus y P. Grünbacher Eds., "Value-Based Software Engineering." Berlin, Heidelberg: Springer-Verlag, 2006. doi: 10.1007/3-540-29263-2.
- [153] M. Fowler, *Patterns of enterprise applications architecture* (The Addison-Wesley Signature Series). Boston, MA, USA: Addison-Wesley, 2003.
- [154] S. J. Mellor, A. N. Clark y T. Futagami, "Model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 14-18, 2003. doi: 10.1109/MS.2003.1231145.
- [155] J. García Molina, F. O. García Rubio, V. Pelechano, A. Vallecillo, J. M. Vara y C. Vicente-Chicote, *Desarrollo de software dirigido por modelos: Conceptos, métodos y herramientas*. Madrid, España: Ra-Ma, 2013.
- [156] D. C. Schmidt, "Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25-31, 2006. doi: 10.1109/MC.2006.58.
- [157] J. Miller y J. Mukerji, "Model Driven Architecture (MDA)," Object Management Group, Needham, MA, USA, ormsc/01-07-01, 2001. Disponible en: <https://goo.gl/CRLSnH>.
- [158] Object Management Group, "Model Driven Architecture (MDA). MDA Guide rev. 2.0," Object Management Group, Needham, MA, USA, ormsc/2014-06-01, 2014. Disponible en: <https://goo.gl/wXwCTo>.
- [159] B. W. Boehm, "Some future trends and implications for systems and software engineering processes," *Systems Engineering*, vol. 9, no. 1, pp. 1-19, 2006. doi: 10.1002/sys.20044.

- [160] M. B. Chrissis, M. Konrad y S. Shrum, *CMMI for development: Guidelines for process integration and product improvement*, 3rd ed. (The SEI Series in Software Engineering). Upper Saddle River, NJ, USA: Addison Wesley, 2011.
- [161] B. W. Boehm, "Anchoring the software process," *IEEE Software*, vol. 13, no. 4, pp. 73-82, 1996. doi: 10.1109/52.526834.
- [162] P. Kroll y P. Kruchten, *The rational unified process made easy: A practitioner's guide to the rational unified process* (Addison-Wesley Object Technology Series). Boston, MA, USA: Addison Wesley, 2003.
- [163] A. Maslow, *Motivation and personality*. New York, NY, USA: Harper and Row, 1954.
- [164] H. R. Booher, "Introduction: Human systems integration," en *Handbook of human systems integration*, H. R. Booher, Ed. Wiley Series in Systems Engineering and Management, pp. 1-30, New York, NY, USA: John Wiley & Sons Inc., 2003.
- [165] B. W. Boehm y A. Jain, "An initial theory of value-based software engineering," en *Value-Based Software Engineering*, S. Biffl, A. Aurum, B. W. Boehm, H. Erdogmus y P. Grünbacher, Eds. pp. 15-37, Berlin, Heidelberg: Springer-Verlag, 2006. doi: 10.1007/3-540-29263-2.
- [166] A. Fuggetta, "Software process: A roadmap," en *Proceedings of the Conference on The Future of Software Engineering (Limerick, Ireland – June 04 - 11, 2000)*, A. Finkelstein, Ed. pp. 25-34, New York, NY, USA: ACM, 2000. doi: 10.1145/336512.336521.
- [167] J. Garzías, J. Enríquez de Salamanca y E. Irrazábal, *Gestión ágil de proyectos software. Versión 1.1*. Madrid, España: Kybele Consulting, 2012.
- [168] B. W. Boehm y R. Turner, *Balancing agility and discipline. A guide for the perplexed*. Boston, MA, USA: Addison Wesley, 2004.
- [169] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, F. J. García-Peñalvo y E. Tovar-Caro, "Project managers in global software development teams: A study of the effects on productivity and performance," *Software Quality Journal*, vol. 22, no. 1, pp. 3-19, 2014. doi: 10.1007/s11219-012-9191-x.
- [170] R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, S. Misra y F. J. García-Peñalvo, "Analyzing Human Resource Management Practices Within the GSD Context," *Journal of Global Information Technology Management*, vol. 15, no. 3, pp. 30-54, 2012. doi: 10.1080/1097198X.2012.10845617.
- [171] M. Alier Forment, M. J. Casany Guerrero, M. Á. Conde González, F. J. García-Peñalvo y C. Severance, "Interoperability for LMS: the missing piece to become the common place for e-learning innovation," *International Journal of Knowledge and Learning (IJKL)*, vol. 6, no. 2/3, pp. 130-141, 2010. doi: 10.1504/IJKL.2010.034749.
- [172] K. Manikas y K. M. Hansen, "Software ecosystems – A systematic literature review," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1294-1306, 2013. doi: 10.1016/j.jss.2012.12.026.
- [173] T. Mens, M. Claes, P. Grosjean y A. Serebrenik, "Studying evolving software ecosystems based on ecological models," en *Evolving Software Systems*, T. Mens, A. Serebrenik y A. Cleve, Eds. pp. 297-326, Berlin, Heidelberg: Springer, 2014. doi: 10.1007/978-3-642-45398-4_10.
- [174] F. J. García-Peñalvo, "Technological Ecosystems for Enhancing the Interoperability and Data Flows," *Journal of Information Technology Research*, vol. 11, no. 1, pp. vi-x, 2018.
- [175] J. Gubbi, R. Buyya, S. Marusic y M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation*

- Computer Systems*, vol. 29, no. 7, pp. 1645-1660, 2013. doi: 10.1016/j.future.2013.01.010.
- [176] L. Atzori, A. Iera y G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787-2805, 2010. doi: 10.1016/j.comnet.2010.05.010.
- [177] M. Brettel, N. Friederichsen, M. Keller y M. Rosenberg, "How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 perspective " *International Journal of Information and Communication Engineering*, vol. 8, no. 1, pp. 37-44, 2014.
- [178] V. Roblek, M. Meško y A. Krapež, "A complex view of Industry 4.0," *SAGE Open*, vol. 6, no. 2, 2016. doi: 10.1177/2158244016653987.
- [179] W. van der Hoek y M. Wooldridge, "Multi-agent systems," en *Handbook of knowledge representation*, F. van Harmelen, V. Lifschitz y B. Porter, Eds. Foundations of Artificial Intelligence, pp. 887-928, Amsterdam, The Netherlands: Elsevier, 2008. doi: 10.1016/S1574-6526(07)03024-6.
- [180] F. J. García-Peñalvo, J. Cruz-Benito, M. Martín-González, A. Vázquez-Ingelmo, J. C. Sánchez-Prieto y R. Therón, "Proposing a Machine Learning Approach to Analyze and Predict Employment and its Factors," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. In Press, 2018. doi: 10.9781/ijimai.2018.02.002.
- [181] A. Cavalcanti, "Assembly automation with evolutionary nanorobots and sensor-based control applied to nanomedicine," *IEEE Transactions on Nanotechnology*, vol. 2, no. 2, pp. 82-87, 2003. doi: 10.1109/TNANO.2003.812590.
- [182] A. Finkelsteini y J. Kramer, "Software engineering: A roadmap," en *Proceedings of the Conference on The Future of Software Engineering (Limerick, Ireland – June 04 - 11, 2000)*, A. Finkelstein, Ed. pp. 3-22, New York, NY, USA: ACM, 2000. doi: 10.1145/336512.336519.
- [183] A. Abran, J. W. Moore, P. Bourque, R. Dupuis y L. L. Tripp, *Guide to the Software Engineering Body of Knowledge SWEBOK*. Los Alamitos, CA, USA: IEEE Computer Society, 2001.
- [184] A. Abran, J. W. Moore, P. Bourque y R. Dupuis Eds., "Guide to the Software Engineering Body of Knowledge. 2004 Version." Los Alamitos, CA, USA: IEEE Computer Society, 2004.
- [185] P. Bourque y R. E. Fairley Eds., "Guide to the Software Engineering Body of Knowledge. Version 3.0. SWEBOK®." USA: IEEE, 2014. Disponible en: <https://goo.gl/UphKi1>.
- [186] T. B. Hilburn, S. Mengel, D. J. Bagert y D. Oexmann, "Software engineering across computing curricula," *ACM SIGCSE Bulletin*, vol. 30, no. 3, pp. 117-121, 1998. doi: 10.1145/290320.283086.
- [187] D. J. Bagert, T. B. Hilburn, G. Hislop, M. Lutz, M. McCracken y S. Mengel, "Guidelines for Software Engineering Education. Version 1.0," Working Group on Software Engineering Education and Training (WGSEET), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Technical Report, CMU/SEI-99-TR-032, 1999.
- [188] T. B. Hilburn, I. Hirmanpour, S. Khajenoori, R. Turner y A. Qasem, "A Software Engineering Body of Knowledge Version 1.0," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Technical Report, CMU/SEI-99-TR-004 (ESC-TR-99-004), 1999.
- [189] P. Bourque, R. Dupuis, A. Abran, J. W. Moore y L. Tripp, "The guide to the Software Engineering Body of Knowledge," *IEEE Software*, vol. 16, no. 6, pp. 35-44, 1999. doi: 10.1109/52.805471.

- [190] S. Sánchez Alonso, M. Á. Sicilia Urbán y D. Rodríguez García, *Ingeniería del Software. Un enfoque desde la guía SWEBOK*. Madrid, España: Ibergarceta Publicaciones, 2011.
- [191] Universidad de Salamanca. (2017). *Guía Académica de la asignatura Ingeniería de Software I. Curso 2017-2018*. Facultad de Ciencias, Departamento de Informática y Automática. Salamanca, España: Universidad de Salamanca. Disponible: <https://goo.gl/nhfAHC>.
- [192] Universidad de Salamanca. (2017). *Guía Académica de la asignatura Ingeniería de Software II. Curso 2017-2018*. Facultad de Ciencias, Departamento de Informática y Automática. Salamanca, España: Universidad de Salamanca. Disponible: <https://goo.gl/Lg3qw3>.
- [193] Universidad de Salamanca. (2017). *Guía Académica de la asignatura Gestión de Proyectos. Curso 2017-2018*. Facultad de Ciencias, Departamento de Informática y Automática. Salamanca, España: Universidad de Salamanca. Disponible: <https://goo.gl/DM5Nam>.
- [194] Gobierno de España. (2009). *Resolución de 8 de junio de 2009, de la Secretaría General de Universidades, por la que se da publicidad al Acuerdo del Consejo de Universidades, por el que se establecen recomendaciones para la propuesta por las universidades de memorias de solicitud de títulos oficiales en los ámbitos de la Ingeniería Informática, Ingeniería Técnica Informática e Ingeniería Química*. Ministerio de Educación. BOE-A-2009-12977, no. 187, de 4 de agosto de 2009, sección III. Otras disposiciones, pp. 66699-66710. Madrid, España: Agencia Estatal Boletín Oficial del Estado. Disponible: <https://goo.gl/fkHSFK>.
- [195] Universidad de Salamanca. (2017). *Guía Académica del Máster Universitario en Ingeniería Informática. Curso 2017-2018*. Facultad de Ciencias. Salamanca, España: Universidad de Salamanca. Disponible: <https://goo.gl/Ly81WG>.
- [196] Universidad de Salamanca. (2017). *Guía Académica del Máster Universitario en Sistemas Inteligentes. Curso 2017-2018*. Facultad de Ciencias. Salamanca, España: Universidad de Salamanca. Disponible: <https://goo.gl/BB8F19>.
- [197] F. J. García-Peñalvo, "Aportaciones de la Ingeniería en una Perspectiva Multicultural de la Sociedad del Conocimiento," *VAEP-RITA*, vol. 1, no. 4, pp. 201-202, 2013.
- [198] F. J. García-Peñalvo, "Education in knowledge society: A new PhD programme approach," en *Proceedings of the First International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'13) (Salamanca, Spain, November 14-15, 2013)*, F. J. García-Peñalvo, Ed. ACM International Conference Proceeding Series (ICPS), pp. 575-577, New York, NY, USA: ACM, 2013. doi: 10.1145/2536536.2536624.
- [199] F. J. García-Peñalvo, "Formación en la sociedad del conocimiento, un programa de doctorado con una perspectiva interdisciplinar," *Education in the Knowledge Society*, vol. 15, no. 1, pp. 4-9, 2014.
- [200] F. J. García-Peñalvo, "Engineering contributions to a Knowledge Society multicultural perspective," *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje (IEEE RITA)*, vol. 10, no. 1, pp. 17-18, 2015. doi: 10.1109/RITA.2015.2391371.
- [201] F. Llopis Pascual y F. Llorens Largo Eds., "Adecuación del primer curso de los estudios de informática al Espacio Europeo de Educación Superior," Serie Docencia Universitaria - EEES. Alcoy, Alicante, España: Marfil, 2005.
- [202] F. J. García-Peñalvo, "Memoria de resultados del Proyecto de Innovación Docente US14/04. Realización de proyectos docentes para asignaturas de Ingeniería Informática bajo las Directrices del Espacio Europeo de Educación

- Superior," Universidad de Salamanca, Salamanca, España, 2006. Disponible en: <https://goo.gl/HkacM1>.
- [203] J. Calvo Gallego y F. J. García-Peñalvo, "Guía Docente de la Asignatura Ingeniería del Software II dentro del Marco del EEES," en *Actas de las I Jornadas de Innovación Educativa de la Escuela Politécnica Superior de Zamora. Las Enseñanzas Técnicas ante el Reto del Espacio Europeo de Educación Superior (Zamora, 20-22 de junio de 2006)*, J. L. Pérez Iglesias, M. L. Pérez Delgado, M. P. Rubio Cavero, J. C. Matos Franco y J. Calvo Gallego, Eds., Zamora, España: Escuela Politécnica Superior de Zamora, 2006.
- [204] F. J. García-Peñalvo y A. García-Holgado, "Introducción a la Ingeniería del Software," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/9bVcWo>. doi: 10.5281/zenodo.1182457.
- [205] C. Larman, *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al Proceso Unificado*, 2ª ed. Madrid, España: Pearson Educación, 2003.
- [206] C. Larman, *Applying UML and patterns. An introduction to object-oriented analysis and design and the Unified Process*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004.
- [207] S. L. Pfleeger, *Ingeniería del Software. Teoría y Práctica*. Argentina: Prentice Hall, 2002.
- [208] R. S. Pressman, *Ingeniería del Software: Un Enfoque Práctico*, 7ª ed. México D. F., México: McGraw-Hill, 2010.
- [209] M. E. Fayad, M. Laitinen y R. P. Ward, "Thinking objectively: Software engineering in the small," *Communications of the ACM*, vol. 43, no. 3, pp. 115-118, 2000. doi: 10.1145/330534.330555.
- [210] A. Fuggetta, "A Classification of CASE Technology," *Computer*, vol. 26, no. 12, pp. 25-38, 1993. doi: 10.1109/2.247645.
- [211] D. Gage, "Consumer products: When software bugs bite," *Baseline. Driving Business Success With Technology*, 2003, Disponible en: <https://goo.gl/BNMvR2>.
- [212] R. L. Glass, "Talk about a software crisis - not!," *Journal of Systems and Software*, vol. 55, no. 1, pp. 1-2, 2000. doi: 10.1016/s0164-1212(00)00043-1.
- [213] J. L. Lions, "ARIANE 5 Flight 501 Failure," Report by the Inquiry Board, 1996. Disponible en: <https://goo.gl/nSH6Ht>.
- [214] L. B. S. Raccoon, "Fifty years of progress in software engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 1, pp. 88-104, 1997. doi: 10.1145/251759.251878.
- [215] Risk Forum. (2018). Disponible en: <https://goo.gl/U9wifz>.
- [216] S. Sharma y A. Rai, "CASE deployment in IS organizations," *Communications of the ACM*, vol. 43, no. 1, pp. 80-88, 2000. doi: 10.1145/323830.323848.
- [217] R. Singh, "The Software Life Cycle Processes standard," *Computer*, vol. 28, no. 11, pp. 89-90, 1995. doi: 10.1109/2.471194.
- [218] E. Yourdon, *Análisis Estructurado Moderno*. México: Prentice-Hall Hispanoamericana, 1993.
- [219] F. J. García-Peñalvo y A. García-Holgado, "Sistemas de información," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/Nk985o>. doi: 10.5281/zenodo.1179280.

- [220] J. Chandra *et al.*, "Information systems frontiers," *Communications of the ACM*, vol. 43, no. 1, pp. 71-79, 2000. doi: 10.1145/323830.323847.
- [221] J. Fernández González, "Business Intelligence: Analizando datos para extraer nueva información y tomar mejores decisiones," *Novática. Revista de la Asociación de Técnicos en Informática*, vol. XXXVII, no. 211, pp. 6-7, 2011.
- [222] A. J. Swartz, "Airport 95: automated baggage system?," *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 2, pp. 79-83, 1996. doi: 10.1145/227531.227544.
- [223] S. White *et al.*, "Systems engineering of computer-based systems," *Computer*, vol. 26, no. 11, pp. 54-65, 1993. doi: 10.1109/2.241426.
- [224] F. J. García-Peñalvo y A. García-Holgado, "Modelos de proceso," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/xYYJGM>. doi: 10.5281/zenodo.1179286.
- [225] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah y R. Madachy, "Using the WinWin spiral model: a case study," *Computer*, vol. 31, no. 7, pp. 33-44, 1998. doi: 10.1109/2.689675.
- [226] I. Gutiérrez y N. Medinilla, "Contra el arraigo de la cascada," en *Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99 (24-26 de noviembre de 1999, Cáceres - España)*, P. Botella, J. Hernández y F. Saltor, Eds. pp. 393-404, 1999.
- [227] B. Henderson-Sellers y J. M. Edwards, "The object-oriented systems life cycle," *Communications of the ACM*, vol. 33, no. 9, pp. 142-159, 1990. doi: 10.1145/83880.84529.
- [228] F. J. García-Peñalvo y A. García-Holgado, "Ingeniería de requisitos," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/jtvuRK>. doi: 10.5281/zenodo.1181464.
- [229] G. Booch, J. Rumbaugh y I. Jacobson, 2ª, Ed. *El lenguaje unificado de modelado* (Object Technology Series). Madrid, España: Pearson Educación, 2007.
- [230] A. Durán y B. Bernárdez, "Metodología para la elicitación de requisitos de sistemas software (versión 2.3)," Universidad de Sevilla, Universidad de Sevilla, España, Informe Técnico LSI-2000-10, 2002. Disponible en: <https://goo.gl/rhV8eV>.
- [231] J. Rumbaugh, I. Jacobson y G. Booch, *El Lenguaje Unificado de Modelado manual de referencia*, 2ª ed. (Object Technology Series). Madrid, España: Pearson Educación, 2007.
- [232] A. Casamayor, D. Godoy y M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436-445, 2010. doi: 10.1016/j.infsof.2009.10.010.
- [233] C. Ebert, "Putting requirement management into praxis: Dealing with nonfunctional requirements," *Information and Software Technology*, vol. 40, no. 3, pp. 175-185, 1998. doi: 10.1016/S0950-5849(98)00049-4.
- [234] D. J. Grimshaw y G. W. Draper, "Non-functional requirements analysis: deficiencias in structured methods," *Information and Software Technology*, vol. 43, no. 11, pp. 629-634, 2001. doi: 10.1016/S0950-5849(01)00171-9.
- [235] A. M. Hickey y A. M. Davis, "The Role of Requirements Elicitation Techniques in Achieving Software Quality," presentado en Eighth International Workshop

- on Requirements Engineering: Foundation for Software Quality, REFSQ'2002 (September 09-10th, 2002), Essen, Germany, 2002.
- [236] L. A. Maciaszek, *Requirements analysis and system design: Developing information systems with UML*. Essex, UK: Addison-Wesley Longman Ltd., 2001.
- [237] P.-W. Ng, "Adopting use cases, Part 1: Understanding types of use cases and artifacts," *IBM developerWorks*: IBM, 2003, Disponible en: <https://goo.gl/2MdMkP>.
- [238] N. Power, "Variety and quality in requirements documentation," presentado en Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ'2001 (June 4-5, 2001), Interlaken, Switzerland, 2001.
- [239] F. J. García-Peñalvo y A. García-Holgado, "Introducción al Proceso Unificado," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/ikvtYC>. doi: 10.5281/zenodo.1181693.
- [240] I. Jacobson, G. Booch y J. Rumbaugh, *El Proceso Unificado de desarrollo de software* (Object Technology Series). Madrid, España: Pearson Educación, 2000.
- [241] P. B. Kruchten, "The 4+1 View Model of architecture," *IEEE Software*, vol. 12, no. 6, pp. 42-50, 1995. doi: 10.1109/52.469759.
- [242] Rational Software, "Rational Unified Process. Best practices for software development teams," Rational Software, Cupertino, CA, USA, Rational Software White Paper, TP026B, Rev 11/01, 1998. Disponible en: <https://goo.gl/5KNng4>.
- [243] F. J. García-Peñalvo y A. García-Holgado, "Flujos de trabajo del Proceso Unificado," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/efbVuV>. doi: 10.5281/zenodo.1181733.
- [244] F. J. García-Peñalvo y A. García-Holgado, "Análisis orientado a objetos," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/LZ3K6z>. doi: 10.5281/zenodo.1181820.
- [245] B. Bruegge y A. H. Dutoit, *Object-oriented software engineering. Using UML, patterns, and Java*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.
- [246] J. J. Odell, *Advanced object-oriented analysis and design using UML* (SIGS Reference Library). SIGS Books & Multimedia, 1998.
- [247] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy y W. Lorensen, *Modelado y diseño orientados a objetos. Metodología OMT*. Hertfordshire, UK: Prentice-Hall International, 1996.
- [248] F. J. García-Peñalvo, M. N. Moreno García y A. García-Holgado, "UML. Unified Modeling Language," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/RV7tY7>. doi: 10.5281/zenodo.1181839.
- [249] F. J. García-Peñalvo y M. N. Moreno, "Software Modeling Teaching in a First Software Engineering Course. A Workshop-Based Approach," *IEEE Transactions on Education*, vol. 47, no. 2, pp. 180-187, 2004. doi: 10.1109/TE.2004.824839.

- [250] F. J. García-Peñalvo, H. Alarcón y Á. Domínguez, "Active learning experiences in Engineering Education," *International Journal of Engineering Education*, vol. In Press, 2019.
- [251] L. D. Glasserman Morales, "Aprendizaje activo en ambientes enriquecidos con tecnología," Doctor en Innovación Educativa, Escuela de Graduados en Educación, Tecnológico de Monterrey, Monterrey, Nuevo León, México, 2013. Disponible en: <https://goo.gl/q4J5Bg>.
- [252] F. J. García-Peñalvo y A. García-Holgado, "Software," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/TR2AZx>. doi: 10.5281/zenodo.1182466.
- [253] F. J. García-Peñalvo y A. García-Holgado, "Concepto de Ingeniería del Software," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/ueF2Cw>. doi: 10.5281/zenodo.1182469.
- [254] F. J. García-Peñalvo y A. García-Holgado, "Proceso," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/gn86Cq>. doi: 10.5281/zenodo.1182475.
- [255] F. J. García-Peñalvo y A. García-Holgado, "Metodologías de Ingeniería de Software," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/qMS6n1>. doi: 10.5281/zenodo.1182491.
- [256] F. J. García-Peñalvo y A. García-Holgado, "Requisitos," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/ordojd>. doi: 10.5281/zenodo.1182551.
- [257] F. J. García-Peñalvo y A. García-Holgado, "Aspectos prácticos de los casos de uso," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/7xUr32>. doi: 10.5281/zenodo.1182585.
- [258] F. J. García-Peñalvo y A. García-Holgado, "Proceso Unificado," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/26jt4t>. doi: 10.5281/zenodo.1182652.
- [259] F. J. García-Peñalvo y A. García-Holgado, "Modelo de Dominio," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/W7CcFH>. doi: 10.5281/zenodo.1182667.
- [260] F. J. García-Peñalvo y A. García-Holgado, "Introducción al análisis orientado a objetos," Recursos docentes de la asignatura Ingeniería de Software I. Grado en

- Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/CCAqN2>. doi: 10.5281/zenodo.1182671.
- [261] F. J. García-Peñalvo y A. García-Holgado, "Fundamentos de la vista de casos de uso," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/py6Bv2>. doi: 10.5281/zenodo.1182682.
- [262] F. J. García-Peñalvo y A. García-Holgado, "Fundamentos de la vista estática," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/5wuad9>. doi: 10.5281/zenodo.1182696.
- [263] F. J. García-Peñalvo y A. García-Holgado, "Fundamentos de la vista de interacción," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/eZsxHq>. doi: 10.5281/zenodo.1182702.
- [264] F. J. García-Peñalvo y A. García-Holgado, "La mujer y la niña en la Ciencia y la Tecnología. Práctica obligatoria. Ingeniería de Software I. Curso 2017-2018," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/9AcUZu>. doi: 10.5281/zenodo.1183299.
- [265] F. J. García-Peñalvo y A. García-Holgado, "Indicaciones para el desarrollo del trabajo final de Ingeniería de Software I. Curso 2017-2018," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018, F. J. García-Peñalvo y A. García-Holgado, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/XseqKa>. doi: 10.5281/zenodo.1183305.
- [266] A. García-Holgado, F. J. García-Peñalvo, J. J. Mena Marcos y C. González González, "Inclusión de la perspectiva de género en la asignatura de Ingeniería de Software I," en "Memorias de Innovación Docente," núm. ID2016/084, Universidad de Salamanca, Salamanca, España, 2017. Disponible en: <https://goo.gl/PzZeAN>.
- [267] A. García-Holgado, F. J. García-Peñalvo, J. J. Mena Marcos y C. González, "Introducción de la Perspectiva de Género en la docencia de Ingeniería del Software," en *La innovación docente como misión del profesorado. Actas del IV Congreso Internacional sobre Aprendizaje, Innovación y Competitividad. CINAIC 2017 (4-6 de Octubre de 2017, Zaragoza, España)*, M. L. Sein-Echaluce Laclela, Á. Fidalgo-Blanco y F. J. García-Peñalvo, Eds. pp. 627-631, Zaragoza, España: Servicio de Publicaciones Universidad de Zaragoza, 2017. doi: 10.26754/CINAIC.2017.000001_134.
- [268] A. García-Holgado, F. J. García-Peñalvo, J. J. Mena Marcos y C. S. González, "Pretest y postest para evaluar la introducción de la perspectiva de género en la docencia de asignaturas de Ingeniería Informática," Grupo GRIAL, Universidad de Salamanca, Salamanca, España, Technical Report, GRIAL-TR-2017-005, 2017. Disponible en: <https://goo.gl/R7Ybr7>. doi: 10.5281/zenodo.825768.

- [269] A. García-Holgado, J. J. Mena Marcos, F. J. García-Peñalvo y C. González, "Inclusion of gender perspective in Computer Engineering careers. Elaboration of a questionnaire to assess the gender gap in Tertiary Education," presentado en EDUCON2018 – IEEE Global Engineering Education Conference (17-20 April 2018), Santa Cruz de Tenerife, Canary Islands, Spain, 2018.
- [270] D. Nachbar, "Bringing real-world software development into the classroom: a proposed role for public software in computer science education," *ACM SIGCSE Bulletin*, vol. 30, no. 1, pp. 171-175, 1998. doi: 10.1145/274790.273753.
- [271] A. García-Holgado, F. J. García-Peñalvo y M. J. Rodríguez-Conde, "Pilot experience applying an active learning methodology in a Software Engineering classroom," presentado en EDUCON2018 – IEEE Global Engineering Education Conference (17-20 April 2018), Santa Cruz de Tenerife, Canary Islands, Spain, 2018.
- [272] F. J. García-Peñalvo, "The WYRED Project: A Technological Platform for a Generative Research and Dialogue about Youth Perspectives and Interests in Digital Society," *Journal of Information Technology Research*, vol. 9, no. 4, pp. vi-x, 2016.
- [273] F. J. García-Peñalvo, "WYRED Project," *Education in the Knowledge Society*, vol. 18, no. 3, pp. 7-14, 2017. doi: 10.14201/eks2017183714.
- [274] F. J. García-Peñalvo y N. A. Kearney, "Networked youth research for empowerment in digital society. The WYRED project," en *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'16) (Salamanca, Spain, November 2-4, 2016)*, F. J. García-Peñalvo, Ed. pp. 3-9, New York, NY, USA: ACM, 2016. doi: 10.1145/3012430.3012489.
- [275] F. J. García-Peñalvo y A. García-Holgado, *Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2017-2018*, Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2018. [Online]. Disponible en: <https://goo.gl/UKhFgc>. doi: 10.5281/zenodo.1182722.
- [276] M. S. Ramírez-Montoya, F. J. García-Peñalvo y R. McGreal, "Shared Science and Knowledge. Open Access, Technology and Education," *Comunicar*, vol. 26, no. 54, pp. 1-5, 2018.
- [277] E. Gamma, R. Helm, R. Johnson y J. Vlissides, *Patrones de Diseño. Elementos de software orientado a objetos reutilizable*. Madrid, España: Pearson Educación, 2003.
- [278] J. Garzás, *Peopleware y equipos ágiles con prácticas de management 3.0*. Madrid, España: 233 grados de TI, 2017.
- [279] B. Meyer, *Construcción de software orientado a objetos*, 2ª ed. Madrid, España: Prentice Hall, 1999.
- [280] S. R. Schach, *Ingeniería de Software clásica y orientada a objetos*, 6ª ed. México: McGraw-Hill, 2006.
- [281] A. Durán y B. Bernárdez, "Metodología para el análisis de requisitos de sistemas software (versión 2.2)," Universidad de Sevilla, Universidad de Sevilla, España, 2001. Disponible en: <https://goo.gl/yuGje1>.
- [282] K. Pohl, "Requirements Engineering: An Overview," en *Encyclopedia of Computer Science and Technology*, vol. 36, A. Kent y J. Williams, Eds., New York, USA: Marcel Dekker, 1997.
- [283] F. J. García-Peñalvo, S. Bravo, M. Á. Conde y H. Barbosa, "SET, A CASE Tool to Guide the Creation of Domain and Use Case Models in an Introductory Software Engineering Course," *International Journal of Engineering Education (IJEE)*, vol. 27, no. 1, pp. 31-40, 2011.

- [284] M. J. Rodríguez-Conde, F. J. García-Peñalvo y A. García-Holgado, "Pretest y postest para evaluar la implementación de una metodología activa en la docencia de Ingeniería del Software," Grupo GRIAL, Universidad de Salamanca, Salamanca, España, Technical Report, GRIAL-TR-2017-007, 2017. Disponible en: <https://goo.gl/wjpf0F>. doi: 10.5281/zenodo.1011121.
- [285] B. Gargallo, J. M. Suárez-Rodríguez y C. Pérez-Pérez, "El cuestionario CEVEAPEU. Un instrumento para la evaluación de las estrategias de aprendizaje de los estudiantes universitarios," *Revista ELección de Investigación y Evaluación Educativa - RELIEVE*, vol. 15, no. 2, pp. 1-31, 2009.
- [286] A. B. González-Rogado, "Evaluación del impacto de una metodología docente, basada en el aprendizaje activo del estudiante, en computación en ingenierías," PhD, Departamento de Didáctica, Organización y Métodos de Investigación. Facultad de Educación, Universidad de Salamanca, Salamanca, España, 2012. Disponible en: <https://goo.gl/ULFdo4>.
- [287] F. J. García-Peñalvo, M. N. Moreno García, J. R. García-Bermejo Giner y A. Luis Reboredo, *Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1. Ingeniería Técnica en Informática de Sistemas. Bienio 1999-2001*. Salamanca, España: Departamento de Informática y Automática. Universidad de Salamanca, 2000.
- [288] F. J. García-Peñalvo, M. N. Moreno García, G. González Talaván y Á. M. Moreno Montero, "Plan de Calidad para Asignaturas en Ingenierías Técnicas en Informática," en *Actas del Congreso Nacional de Informática Educativa, CONIED'99 (Puertollano, Ciudad Real, 17-19 de noviembre de 1999)*, M. Ortega y J. Bravo, Eds., 1999.
- [289] F. J. García-Peñalvo, E. Montero García y P. Arranz Val, "Proceso de evaluación por pares. Una experiencia práctica," presentado en II Jornadas de Calidad y Universidad: Calidad en la Docencia (10-11 de noviembre de 1998), Burgos, España, 1998.
- [290] F. J. García-Peñalvo, E. Montero García y P. Arranz Val, "Protocolo de evaluación por pares," en *Actas del VII Congreso Iberoamericano de Educación Superior en Computación - CIESC'99 (Asunción-Paraguay, 29 agosto - 3 septiembre de 1999)* pp. 38-47, 1999.
- [291] F. J. García-Peñalvo, M. N. Moreno García, A. M. Moreno, G. González, B. Curto y F. J. Blanco, "ADAM CASE: Using upper CASE tools in Software Engineering laboratory," en *Computers and Education: Towards an Interconnected Society*, M. Ortega y J. Bravo, Eds. pp. 149-158, The Netherlands: Kluwer Academic Publishers, 2001. doi: 10.1007/0-306-47533-2_14.
- [292] F. J. García-Peñalvo y I. Álvarez, "Left CASE – A free software component-based CASE tool for software engineering practice support," en *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference, FIE 2003 (Boulder, Colorado, USA, November 5-8, 2003)*, vol. 3 pp. S2C-7-S2C-12, USA: IEEE, 2003. doi: 10.1109/FIE.2003.1265945.
- [293] F. J. García-Peñalvo y M. N. Moreno, "C-requirements specification teaching," en *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference, FIE 2003 (November 5-8, 2003, Boulder, Colorado, USA)*, vol. 3 pp. S2C-1-S2C-6, USA: IEEE, 2003. doi: 10.1109/FIE.2003.1265944.
- [294] F. J. García Peñalvo, "Pensando en ECTS. Un Caso Práctico para la Asignatura de Ingeniería del Software," en *Los Estudios de Ingeniería Informática en el Espacio Europeo de Educación Superior. Contexto y Realidad en la Comunidad Autónoma de Castilla y León*, F. J. García Peñalvo, Ed. Aquilafuente, no. 101, pp. 139-156, Salamanca, España: Ediciones Universidad de Salamanca, 2006.

- [295] F. J. García-Peñalvo, S. Bravo-Martín y M. Á. Conde-González, *12522 - Ingeniería del Software, 2008-09*, Salamanca, España: Portal OCW. Universidad de Salamanca, 2008. [Online]. Disponible en: <https://goo.gl/9sH4Zx>.
- [296] F. J. García-Peñalvo, J. Cruz-Benito, M. Á. Conde y D. Griffiths, "Virtual placements for informatics students in open source business across Europe," en *2014 IEEE Frontiers in Education Conference Proceedings (October 22-25, 2014 Madrid, Spain)* pp. 2551-2555, USA: IEEE, 2014. doi: 10.1109/FIE.2014.7044411.
- [297] F. J. García-Peñalvo, J. Cruz-Benito, M. Á. Conde y D. Griffiths, "Semester of Code: Piloting Virtual Placements for Informatics across Europe," en *Proceedings of Global Engineering Education Conference, EDUCON 2015. Tallinn, Estonia, 18-20 March 2015* pp. 567-576, USA: IEEE, 2015. doi: 10.1109/EDUCON.2015.7096026.
- [298] F. J. García-Peñalvo, J. Cruz-Benito, D. Griffiths y A. P. Achilleos, "Virtual placements management process supported by technology: Proposal and firsts results of the Semester of Code," *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje (IEEE RITA)*, vol. 11, no. 1, pp. 47-54, 2016. doi: 10.1109/RITA.2016.2518461.
- [299] D. A. Gómez-Aguilar, R. Therón y F. J. García-Peñalvo, "Understanding educational relationships in Moodle with ViMoodle," en *Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies, 2008. ICALT '08. Santander, Cantabria, Spain, 1-5 July 2008*, P. Díaz, A. Ignacio y E. Mora, Eds. pp. 954-956, USA: IEEE, 2008. doi: 10.1109/ICALT.2008.276.
- [300] D. A. Gómez-Aguilar, M. Á. Conde-González, R. Therón y F. J. García-Peñalvo, "Retrieval information model for Moodle data visualization," en *Proceedings of the 10th IEEE International Conference on Advanced Learning Technologies (ICALT 2010). (Sousse, Tunisia, July 5-7, 2010)* pp. 526-527, USA: IEEE, 2010. doi: 10.1109/ICALT.2010.150.
- [301] D. A. Gómez-Aguilar, M. Á. Conde-González, R. Therón y F. J. García-Peñalvo, "Reveling the evolution of semantic content through visual analysis," en *Proceedings of the 11th IEEE International Conference on Advanced Learning Technologies (ICALT 2011). (Athens, Georgia, USA, July 6-8, 2011)* pp. 450-454, USA: IEEE, 2011. doi: 10.1109/ICALT.2011.141.
- [302] D. A. Gómez Aguilar, M. Á. Conde-González, R. Therón y F. J. García-Peñalvo, "Supporting Moodle-based lesson of Software Engineering through visual analysis," en *Actas del XII Congreso Internacional de Interacción Persona-Ordenador - Interacción 2011. (Lisboa, Portugal, 2-6 de septiembre de 2011)*, N. Garay y J. Abascal, Eds. pp. 319-328, Madrid, España: Ibergarceta Publicaciones S.L., 2011.
- [303] D. A. Gómez Aguilar, M. Á. Conde-González, R. Therón y F. J. García-Peñalvo, "Supporting Moodle-based lesson through visual analysis," en *Human-Computer Interaction - INTERACT 2011, 13th IFIP TC 13 International Conference Proceedings, Part IV (Lisboa, Portugal, September 5-9, 2011)*, P. Campos, J. Jorge, N. Nunes, P. Palanque y M. Winckler, Eds. Lecture Notes in Computer Science, no. LNCS 6949, pp. 604-607, Berlin: Springer, 2011. doi: 10.1007/978-3-642-23768-3.
- [304] D. A. Gómez-Aguilar, F. J. García-Peñalvo y R. Therón, "Evaluación visual de las relaciones entre participación de los estudiantes y sus resultados en entornos de e-learning," en *Actas del XV Simposio Internacional de Tecnologías de la Información y las Comunicaciones en la Educación, SINTICE 2013. Celebrado conjuntamente con el Congreso Español de Informática (CEDI 2013). (17-20 de septiembre de 2013. Facultad de Informática de la Universidad Complutense de*

- Madrid, España), B. Fernández Manjón, Ed. pp. 153-160, Madrid: Universidad Complutense de Madrid, 2013.
- [305] M. Á. Conde, F. J. García-Peñalvo, D. A. Gómez-Aguilar y R. Therón, "Visual learning analytics techniques applied in software engineering subjects," en *2014 IEEE Frontiers in Education Conference Proceedings (October 22-25, 2014 Madrid, Spain)* pp. 3009-3017, USA: IEEE, 2014. doi: 10.1109/FIE.2014.7044486.
- [306] M. Á. Conde, F. J. García-Peñalvo, D. A. Gómez-Aguilar y R. Therón, "Exploring Software Engineering Subjects by Using Visual Learning Analytics Techniques," *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje (IEEE RITA)*, vol. 10, no. 4, pp. 242-252, 2015. doi: 10.1109/RITA.2015.2486378.
- [307] W. S. Humphrey, *Introducción al Proceso Software Personal*. Madrid, España: Pearson Educación, 2001.
- [308] W. S. Humphrey, *PSP. A self-improvement process for software engineers* (The SEI Series in Software Engineering). Upper Saddle River, NJ, USA: Addison-Wesley, 2005.
- [309] G. L. Huber, "Aprendizaje activo y metodologías educativas," *Revista de Educación*, no. Extraordinario 2008, pp. 59-81, 2008.
- [310] J. Benegas, M. C. Pérez de Landazábal y J. Otero Eds., "El aprendizaje activo de la física básica universitaria." Santiago de Compostela, España: Andavira Editora, 2013.
- [311] L. D. Glasserman, F. J. Mortera y M. S. Ramírez-Montoya, "Caracterizando recursos educativos abiertos (REA) y objetos de aprendizaje (OA) que fomentan un aprendizaje activo en los alumnos de primaria," en *Conexión de repositorios educativos digitales: Educonector.info*, F. J. Mortera y M. S. Ramírez-Montoya, Eds. pp. 26-34, México: Lulú editorial digital, 2013.
- [312] A. Dominguez, G. Zavala y M. E. Truyol, "Teaching mathematics using active learning: Teachers' preparation in Chile," en *Proceedings of the 124th ASEE Annual Conference and Exposition; Columbus; United States; 25 June 2017 through 28 June 2017*, vol. 2017-June, USA: ASEE, 2017.
- [313] D. Ramírez, M. S. Ramírez-Montoya y T. M. Marrero, "Novel use of a remote laboratory for active learning in class," *Chemical Engineering Education*, vol. 50, no. 2, pp. 141-148, 2016.
- [314] G. Zavala, M. E. Truyol y A. Dominguez, "Professional development program on active learning for engineering faculty in Chile: First stage," en *Proceedings of the 124th ASEE Annual Conference and Exposition; Columbus; United States; 25 June 2017 through 28 June 2017*, vol. 2017-June, USA: ASEE, 2017.