Opositando

Tema 44: Análisis y diseño de aplicaciones informáticas

27 de septiembre de 2026

Síguenos en Instagram y entérate de nuestras últimas actualizaciones:

opositando.shop

Índice

1. Introducción 1
2. Enfoque teórico del análisis y diseño de aplicaciones informáticas 2
   - 2.1 Conceptos fundamentales: análisis y diseño 2
   - 2.2 Modelos y metodologías en el desarrollo de software 3
   - 2.3 Ciclo de vida del software y fases principales 4
   - 2.4 Herramientas y lenguajes de modelado 5
   - 2.5 Técnicas de recogida y análisis de requisitos 6
   - 2.6 Técnicas de diseño estructurado y orientado a objetos 7
   - 2.7 Integración del diseño y la arquitectura de software 8
3. Implicaciones didácticas en el contexto educativo 9
4. Conclusión 11
5. Bibliografía 12

1. Introducción

El **análisis y diseño de aplicaciones informáticas** constituye una disciplina esencial en el ámbito de la informática, especialmente en los estudios de Formación Profesional y, de forma transversal, en la consolidación de competencias profesionales dentro del sector tecnológico. Atender a la correcta comprensión y dominio de estos procesos es imprescindible ante los cambios constantes y acelerados en el desarrollo de soluciones tecnológicas.

Esta temática resulta estratégica debido a que las aplicaciones informáticas forman la base operativa de organizaciones, empresas e instituciones, incidiendo directamente en la eficiencia, la innovación y la competitividad. Por tanto, analizar y diseñar aplicaciones de manera sistemática y rigurosa permite asegurar la adecuación técnica, funcional y estructural de dichas soluciones respecto a las necesidades reales del contexto en el que se implantan.

El enfoque didáctico de este tema pretende no solo transmitir marcos teóricos, sino también favorecer la adquisición de capacidades críticas y procedimentales necesarias para que el alumnado se integre eficazmente en equipos multidisciplinares y pueda incorporar las mejores prácticas reconocidas en el sector. Según **Sommerville (2016)**, el éxito de proyectos informáticos depende en gran medida de la calidad y robustez de sus fases de análisis y diseño, así como de la interacción equilibrada entre los actores implicados.

En el marco de la **Ley Orgánica 3/2020**, de 29 de diciembre, por la que se modifica la Ley Orgánica 2/2006, de 3 de mayo, de Educación (LOMLOE), y el **Real Decreto 659/2023, de 18 de julio**, que regula la ordenación del sistema de Formación Profesional, se destaca la necesidad de formar perfiles competentes que aborden la resolución de problemas complejos, el desarrollo de propuestas innovadoras y la aplicación de marcos ágiles y adaptativos en la ingeniería del software.

Resulta imprescindible, por tanto, un abordaje académico y actualizado del análisis y el diseño de aplicaciones informáticas, que promueva una capacitación integral y alineada con los estándares profesionales, académicos y legislativos actualmente vigentes. Este enfoque debe incluir tanto el dominio de herramientas y lenguajes de programación como la comprensión de principios metodológicos y buenas prácticas, fomentando competencias que permitan a los profesionales adaptarse a los cambios tecnológicos y responder eficazmente a las demandas del entorno laboral y académico.

2. Enfoque teórico del análisis y diseño de aplicaciones informáticas

2.1 Conceptos fundamentales: análisis y diseño

El proceso de creación de soluciones informáticas precisa, de manera esencial, distinguir y articular los conceptos de **análisis** y **diseño**. Ambos constituyen fases diferenciadas, aunque interdependientes, dentro del paradigma de la ingeniería del software.

El **análisis** hace referencia a la identificación, estudio y estructuración de las necesidades y requisitos que debe satisfacer un sistema informático. En este sentido, el análisis traduce un problema del contexto real en una visión abstracta, descomponiendo esa problemática en partes más manejables que puedan ser interpretadas por equipos de desarrollo y stakeholders. Según **Sommerville (2016)**, el análisis supone entender "qué" debe hacer el sistema, dejando para fases ulteriores la determinación de "cómo" hacerlo.

Por otro lado, el **diseño** se orienta a la elaboración de modelos y arquitecturas que permitan materializar los requisitos anteriormente identificados. En esta fase, se definen las estructuras, relaciones y mecanismos técnicos que darán forma a la solución. El diseño responde al desafío de convertir una visión conceptual en un conjunto concreto de recursos, componentes y procedimientos tecnológicos.

Durante el proceso de análisis, es habitual la utilización de técnicas de descomposición funcional y definición de casos de uso para asegurar una perspectiva precisa y completa de los requerimientos. En contraste, el diseño recurre a diagramas estructurales, patrones y principios de arquitectura para garantizar la coherencia y viabilidad técnica de la solución.

Por ejemplo, al abordar el desarrollo de un sistema de gestión bibliotecaria, el análisis implicaría entrevistar a usuarios y responsables, documentando necesidades como el préstamo de libros, la consulta de catálogos o la gestión de usuarios; mientras que en la fase de diseño se definirían los módulos software, la lógica de negocio, las interfaces gráficas y la estructura de la base de datos que soportará dichas funciones.

La correcta delimitación de ambas etapas resulta determinante para evitar errores de conceptualización y sobrecostes en fases posteriores, tal como evidencian los principios de ingeniería del software recogidos por **Sommerville (2016)**.

2.2 Modelos y metodologías en el desarrollo de software

El desarrollo de aplicaciones informáticas requiere del uso de metodologías estructuradas que permitan organizar, secuenciar y controlar las distintas fases del ciclo de vida de un software. La elección de un modelo de desarrollo resulta decisiva, ya que condiciona la dinámica del equipo, la gestión de los cambios y la relación con el cliente o usuario final.

Entre los modelos más destacados, encontramos el **modelo en cascada**, el **modelo incremental**, el **modelo en espiral** y los **modelos ágiles**. Cada enfoque responde a necesidades y contextos específicos, ajustándose a la complejidad, criticidad y entorno cambiante del proyecto.

El **modelo en cascada** se caracteriza por una secuencialidad rígida de fases, donde no se inicia una etapa sin finalizar la anterior. Este enfoque es adecuado en proyectos con requerimientos muy estables y bien definidos desde el inicio. En la práctica, si se estuviera desarrollando una aplicación para el cálculo de nóminas en la administración pública, y los requisitos legales están perfectamente determinados, el modelo en cascada puede facilitar la trazabilidad y el control documental del proceso.

Por su parte, los **modelos incrementales** permiten iterar el desarrollo, es decir, dividir el sistema en versiones o incrementos funcionales, de modo que cada entrega añade funcionalidades adicionales a la anterior. Esto facilita la obtención de prototipos tempranos que pueden ser evaluados por el cliente.

El **modelo en espiral** combina enfoques iterativos y de gestión del riesgo, resultando útil en proyectos innovadores o de alta incertidumbre técnica, como aquellos asociados a inteligencia artificial o sistemas críticos para la salud, ya que cada ciclo aborda la identificación y tratamiento de riesgos, así como la validación de requisitos.

En la actualidad cobran especial relevancia los **modelos ágiles**, entre los que destaca **Scrum**. Estos modelos priorizan la adaptación al cambio, la entrega continua de valor y la colaboración activa con el cliente. En un entorno educativo, el desarrollo ágil permite al alumnado practicar ciclos cortos de desarrollo (“sprints”) y revisar los resultados periódicamente con sus profesores, asimilando progresivamente los principios de mejora continua y adaptación.

La selección de la metodología más adecuada requiere analizar múltiples factores, como tamaño del equipo, presupuesto, expectativas del cliente, criticidad de la aplicación, nivel de incertidumbre y necesidad de validación rápida.

2.3 Ciclo de vida del software y fases principales

El **ciclo de vida del software** constituye el conjunto ordenado de etapas que recorren las aplicaciones desde su concepción hasta la retirada definitiva. Comprender y gestionar correctamente estas fases es fundamental para asegurar el éxito del proceso de desarrollo y el mantenimiento posterior del producto.

Las fases principales, consensuadas en la literatura especializada (Sommerville, 2016), se pueden sintetizar en las siguientes:

- **Estudio de viabilidad y planificación**: analiza la oportunidad técnica, económica y operativa de producir el software. En un proyecto de digitalización municipal, por ejemplo, esta fase implicaría comparar alternativas, estimar retornos de inversión y estudiar las políticas de privacidad.
- **Análisis de requisitos**: profundiza en las necesidades del usuario, delimitando las funcionalidades, restricciones y objetivos clave del sistema. Un caso práctico puede ser la recogida de especificaciones para una plataforma educativa online, implicando tanto a docentes como a estudiantes para identificar funcionalidades esenciales como foros, evaluaciones y sistemas de mensajería.
- **Diseño**: define la arquitectura, las interfaces y la organización interna del software. Durante el diseño de una aplicación móvil bancaria, se concretarían los módulos de autenticación, transacciones, consulta de saldo y medidas de seguridad.
- **Implementación o codificación**: transforma los modelos diseñados en código ejecutable utilizando lenguajes de programación y herramientas seleccionadas en la fase de diseño.
- **Pruebas y validación**: verifica que el sistema implementa correctamente lo especificado, realizando pruebas unitarias, integradas y de sistema. Así, en una solución de comercio electrónico, se comprobaría que las transacciones, los descuentos y las notificaciones funcionen tal como se definieron.
- **Despliegue y mantenimiento**: implica la instalación, formación de usuarios, resolución de incidencias y actualización del software conforme evolucionan necesidades y tecnologías.

En los procesos reales, estas fases no siempre se dan de forma estrictamente secuencial, existiendo retroalimentaciones entre ellas. Por ejemplo, la detección de un cambio normativo en la fase de pruebas puede requerir reabrir análisis y rediseñar ciertos módulos.

La comprensión integral del ciclo de vida permite a los futuros profesionales evaluar, calendarizar y controlar los recursos de un proyecto informático, facilitando la anticipación de riesgos y la comunicación eficaz con el resto de implicados.

2.4 Herramientas y lenguajes de modelado

El **modelado de software** es una actividad esencial en el análisis y diseño de aplicaciones, ya que posibilita abstraer, documentar y comunicar las características y relaciones de los sistemas. Para ello se utilizan diferentes tipos de herramientas y lenguajes, entre los que destacan los **lenguajes gráficos** tales como los diagramas UML (Unified Modeling Language) y los **lenguajes formales**, que permiten una especificación matemática y precisa de los sistemas.

Los **diagramas de clases**, **diagramas de casos de uso**, **diagramas de secuencia** y **diagramas de actividades** representan, respectivamente, la estructura estática, las funcionalidades, las interacciones temporales y los flujos de trabajo del sistema. Estos modelos son especialmente útiles para lograr una comprensión compartida entre los distintos actores del proyecto.

Por ejemplo, al modelar el sistema de reservas de un teatro, se emplearía un **diagrama de casos de uso** para identificar interacciones clave como "Comprar entrada", "Seleccionar butaca" o "Consultar disponibilidad". Un **diagrama de clases** permitiría especificar las entidades implicadas, como Cliente, Reserva, Evento y Pago, así como sus relaciones.

El uso de herramientas como **Enterprise Architect**, **Visual Paradigm** o **StarUML** facilita la creación, documentación y actualización de estos modelos, ya que permiten generar representaciones automatizadas, mantener la consistencia entre los diagramas y generar documentación técnica. Además, algunos entornos integran la capacidad de generar código a partir de modelos, lo que agiliza la transición del diseño a la implementación.

Debe destacarse también la importancia de los **lenguajes formales** (por ejemplo, Z, VDM) en aplicaciones críticas donde se requiere una verificación exhaustiva de las especificaciones, como ocurre en sistemas ferroviarios o aeroespaciales.

El modelado constituye una actividad iterativa: a medida que se recaba información y surgen nuevos requisitos, los modelos pueden ser actualizados, refinados o ampliados para garantizar su alineación con las necesidades del proyecto.

La capacidad de modelar adecuadamente un sistema es un indicador clave de calidad en el desarrollo software, como subraya **Sommerville (2016)**, ya que reduce ambigüedades y previene errores de comunicación.

2.5 Técnicas de recogida y análisis de requisitos

El análisis de requisitos representa una de las actividades más críticas del desarrollo de aplicaciones, ya que la definición deficiente de los mismos es uno de los principales focos de problemas en proyectos informáticos.

Existen diferentes técnicas y herramientas que permiten identificar, recabar, organizar y analizar los requisitos del usuario o cliente, resultando fundamental una correcta selección y ejecución de las mismas para garantizar una visión comprensiva e integral del problema a resolver.

Entre las técnicas más utilizadas se encuentran:

- **Entrevistas**: consisten en mantener conversaciones estructuradas o semiestructuradas con los usuarios, gestores o técnicos del sistema actual, con el objetivo de identificar sus expectativas y necesidades. Por ejemplo, en el desarrollo de un sistema de gestión académica, se pueden realizar entrevistas a docentes, personal administrativo y alumnado para conocer los procesos actuales y los puntos de mejora.
- **Cuestionarios y encuestas**: permiten recoger información de un número elevado de usuarios potenciales, estructurando las respuestas para facilitar su análisis sistemático. Si se está desarrollando una aplicación para la gestión de citas médicas, el uso de cuestionarios online permite captar la opinión de pacientes acerca de la experiencia actual y sus necesidades tecnológicas.
- **Observación directa**: implica el seguimiento y análisis de los procesos en el entorno real de trabajo, lo que posibilita detectar necesidades no explicitadas por los usuarios. En proyectos industriales, la observación puede revelar cuellos de botella en procesos automatizados que no se comunican verbalmente.
- **Grupos focales**: reúnen a un grupo homogéneo o heterogéneo de personas para discutir necesidades, aportar diferentes puntos de vista y llegar a consensos sobre funcionalidades y prioridades.
- **Análisis de documentación existente**: revisar manuales, informes y sistemas previos permite aprovechar el conocimiento acumulado y evitar la omisión de requisitos críticos. Así, si se va a migrar un sistema de nóminas, analizar la documentación del proceso actual es imprescindible para detectar reglas de negocio no documentadas.

Una vez recogida la información, es necesario estructurar los requisitos en **requisitos funcionales** (qué debe hacer el sistema) y **requisitos no funcionales** (restricciones, rendimiento,

seguridad, escalabilidad, etc.). Según **Sommerville (2016)**, esta distinción ayuda a clarificar los objetivos del software y asegurar la viabilidad técnica de la solución.

Por ejemplo, en el caso de una plataforma de educación a distancia, un requisito funcional sería "Permitir la entrega de actividades evaluables por parte del alumnado", mientras que un requisito no funcional sería "Garantizar un tiempo de respuesta inferior a 3 segundos en horario punta".

La validación de los requisitos implica contrastar los resultados obtenidos con los usuarios para confirmar su corrección y completitud, usando técnicas como la **prototipación rápida** y **revisión de requisitos**.

Una adecuada gestión de requisitos, tal como se recoge en la norma **ISO/IEC/IEEE 29148:2018**, incrementa notablemente las probabilidades de éxito del proyecto y previene desviaciones de alcance, coste y plazo.

2.6 Técnicas de diseño estructurado y orientado a objetos

El diseño de aplicaciones informáticas ha evolucionado a lo largo del tiempo, pasando de técnicas orientadas a la función (diseño estructurado) a enfoques centrados en los objetos (orientados a objetos), en función de la complejidad y naturaleza de los sistemas.

El **diseño estructurado**, popularizado en la década de 1970, utiliza técnicas como los **diagramas de flujo**, los **diagramas de estructura** y los **diagramas de descomposición funcional**. Esta aproximación segmenta el sistema en módulos y procedimientos que se comunican a través de interfaces bien definidas. Resulta especialmente útil en sistemas transaccionales, como programas de gestión de inventarios o contabilidad, donde la secuencialidad y la modularidad permiten una clara organización del código.

Por ejemplo, al diseñar un programa para el cálculo automático de matrículas universitarias, se descomponen las funciones en módulos como "cálculo de tasas", "aplicación de descuentos", y "generación de recibos", representando sus relaciones y dependencias mediante diagramas estructurados.

Por su parte, el **diseño orientado a objetos** se centra en la identificación de **clases**, **objetos**, **atributos** y **métodos** que modelan entidades y comportamientos del mundo real o del dominio de aplicación. Utilizando diagramas UML, este enfoque facilita la reutilización y extensibilidad, permitiendo encapsular datos y responsabilidades. El uso de **herencia**, **polimorfismo** y **abstracción** favorece la creación de sistemas más robustos y adaptables, tal como señala **Sommerville (2016)**.

Siguiendo este paradigma, al diseñar una aplicación para la gestión de envíos postales, se identifican clases como Paquete, Cliente y Envío, definiéndose atributos (peso, dimensiones, dirección de entrega) y métodos (calcular coste, asignar ruta). Las relaciones de herencia permitirán crear subclases específicas, como EnvíoExpress, que heredan de Envío y redefinen algunas de sus funciones para el tratamiento urgente.

El diseño orientado a objetos es especialmente ventajoso en entornos donde los requisitos tienden a evolucionar, ya que los cambios pueden incorporarse aislando las modificaciones en determinadas clases, minimizando el impacto sobre el resto del sistema.

Una adecuada combinación de técnicas estructuradas y orientadas a objetos permite abordar el análisis y diseño en una amplia gama de escenarios, eligiendo en cada caso la aproximación más apropiada en función de la complejidad, estabilidad y dominio del problema.

2.7 Integración del diseño y la arquitectura de software

Una vez definidos los elementos básicos del diseño, se avanza hacia la **integración arquitectónica**, es decir, la definición de cómo los diversos módulos, componentes y servicios interactúan entre sí para conformar una solución coherente, robusta y escalable.

La **arquitectura de software** comprende la organización fundamental de un sistema, sus componentes principales, sus relaciones, y los principios que guían su diseño y evolución, según la definición ampliamente aceptada por IEEE. Establecer una arquitectura clara permite anticipar las implicaciones de escalabilidad, rendimiento, seguridad y mantenibilidad del sistema.

Por ejemplo, en el diseño de una aplicación de comercio electrónico, la arquitectura podría organizarse en una estructura **cliente-servidor**, donde los componentes de interfaz de usuario se comunican con servidores de aplicación, que a su vez acceden a bases de datos para la gestión de productos y pedidos. Definir esta arquitectura implica detallar los protocolos de comunicación, las API entre módulos y los mecanismos de autenticación y gestión de sesiones.

En sistemas distribuidos, la selección de una arquitectura basada en **microservicios** permite dividir el sistema en pequeños servicios independientes que se despliegan, escalan y mantienen de forma autónoma. Así, una plataforma de vídeo bajo demanda puede separar en microservicios la gestión de usuarios, la facturación, el streaming y las recomendaciones, facilitando la adaptación y escalabilidad.

El empleo de **patrones arquitectónicos** reconocidos, como **MVC (Modelo–Vista–Controlador)** para aplicaciones web o **Repositorio** para la gestión de datos, facilita la integración de solucio-

nes robustas y la transferencia de conocimiento entre profesionales. La aplicación disciplinada de estos patrones permite además garantizar la calidad, robustez y sostenibilidad del software, como subrayan **Bass, Clements y Kazman (2012)**.

La integración del diseño y la arquitectura no se limita únicamente a la etapa inicial, sino que debe acompañar a la evolución del sistema, previendo futuros cambios tecnológicos o de negocio. En entornos de Formación Profesional, fomentar el análisis crítico de diferentes arquitecturas, su aplicabilidad y los riesgos asociados contribuye decisivamente al desarrollo del pensamiento analítico y de las competencias profesionales del alumnado.

3. Implicaciones didácticas en el contexto educativo

La incorporación del análisis y el diseño de aplicaciones informáticas en los programas de Formación Profesional responde a la creciente demanda de profesionales que dominen no solo aspectos técnicos, sino también metodológicos, organizativos y transversales, en consonancia con los principios de la **Ley Orgánica 3/2020 (LOMLOE)** y el **Real Decreto 659/2023 de Formación Profesional**.

En este contexto, la enseñanza del análisis y diseño de aplicaciones informáticas promueve el desarrollo de competencias clave como el pensamiento lógico, la capacidad crítica, la creatividad y el trabajo colaborativo. Estas competencias son esenciales para que el alumnado afronte con éxito tareas complejas, participe en equipos multidisciplinares y se adapte a las continuas transformaciones de la tecnología y el mercado laboral.

La LOMLOE establece, en su artículo 43, la importancia de la "integración transversal de las competencias digitales en las distintas materias y módulos, la personalización del aprendizaje y la apuesta decidida por la formación dual y la vinculación con el mundo productivo". Estas consignas se aterrizan en el currículo de los ciclos formativos, especialmente en módulos como "Sistemas y aplicaciones informáticas" y "Desarrollo de aplicaciones multiplataforma" o "Desarrollo de aplicaciones web", donde se aborda de manera práctica el proceso completo de análisis, diseño, implementación y validación de soluciones software.

Por su parte, el **Real Decreto 659/2023** recalca "el fomento del aprendizaje basado en retos, la utilización de metodologías activas y el desarrollo de proyectos integrados". El análisis y diseño de aplicaciones informáticas constituyen el eje vertebrador de esta filosofía, al fomentar que el alumnado realice proyectos donde debe identificar requerimientos, modelar soluciones, elegir arquitecturas, implementar y evaluar su trabajo de forma iterativa.

En términos didácticos, es fundamental conectar el aprendizaje teórico con la experimentación práctica, recurriendo a **casos reales**, trabajo en equipo, utilización de herramientas profesionales y la simulación de entornos empresariales. La utilización de **metodologías ágiles, prototipado rápido y revisión entre pares** permite que el alumnado viva experiencias auténticas, adaptadas al mercado actual.

Además, el aprendizaje de técnicas de análisis y diseño favorece la **inclusión** de alumnado con diferentes estilos y ritmos de aprendizaje, ya que pueden abordarse retos desde enfoques diferentes, facilitando el desarrollo individual y colectivo.

El profesorado debe actuar como guía, facilitando recursos, simulando escenarios reales, fomentando la creatividad y el pensamiento reflexivo, y evaluando no solo los resultados, sino también los procesos y actitudes desarrolladas en el análisis y diseño.

Asimismo, el enfoque de género y la diversidad deben integrarse de manera transversal en la enseñanza del análisis y diseño, visibilizando referentes diversos y promoviendo la igualdad de oportunidades en el acceso y desarrollo profesional en el sector tecnológico.

La integración adecuada de este tema en el currículo prepara, por tanto, al alumnado para participar activamente en la sociedad digital, impulsando la competitividad y la innovación, tal como establecen la **LOMLOE** y el **RD 659/2023**.

La **Ley Orgánica 3/2022, de 31 de marzo, de ordenación e integración de la Formación Profesional**, configura el marco en el que se inscribe esta materia: establece un sistema único de formación profesional, acumulable y acreditable, organizado en grados progresivos y vinculado a los estándares de competencia del Catálogo Nacional. Su aplicación didáctica obliga a orientar la programación hacia resultados de aprendizaje acreditables y a integrar la formación en empresa como parte ordinaria del proceso formativo.

4. Conclusión

El análisis y diseño de aplicaciones informáticas representa un campo de conocimiento fundamental, tanto para la formación integral de los estudiantes de Formación Profesional, como para el fortalecimiento de los procesos de digitalización e innovación en el entorno productivo y social. Su estudio permite desarrollar competencias técnicas y metodológicas esenciales, así como habilidades de resolución de problemas, pensamiento crítico y adaptación a nuevas tecnologías, contribuyendo de manera significativa al avance de proyectos innovadores y a la mejora de la eficiencia en diferentes ámbitos profesionales y comunitarios.

A lo largo de este tema se ha profundizado en los conceptos esenciales de análisis y diseño, los principales modelos y metodologías aplicados, el ciclo de vida del software y la importancia del modelado en la ingeniería informática. Se ha argumentado la relevancia de una adecuada recogida y gestión de requisitos, el uso de técnicas estructuradas y orientadas a objetos, y la conexión imprescindible entre diseño y arquitectura de software para lograr aplicaciones robustas, adaptables y alineadas con las necesidades reales.

Todo ello, acompañado de ejemplos contextualizados y perspectivas didácticas fundamentadas, evidencia la importancia de relacionar estos conocimientos con las competencias profesionales, personales y sociales que demanda la legislación vigente. En definitiva, el dominio del análisis y diseño de aplicaciones informáticas implica no solo saber aplicar técnicas y herramientas, sino también desarrollar una actitud crítica, reflexiva y colaborativa, alineada con los objetivos de la **LOMLOE**, el **Real Decreto 659/2023** y los estándares internacionales en ingeniería y formación TIC.

5. Bibliografía

- Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice*. Addison-Wesley.
- International Organization for Standardization, International Electrotechnical Commission, & Institute of Electrical and Electronics Engineers. (2018). *ISO/IEC/IEEE 29148:2018. Systems and software engineering. Life cycle processes. Requirements engineering*. ISO.
- Ley Orgánica 3/2020, de 29 de diciembre, por la que se modifica la Ley Orgánica 2/2006, de 3 de mayo, de Educación. (2020). *Boletín Oficial del Estado*, 340, de 30 de diciembre de 2020, 122868–122953.
- Ley Orgánica 3/2022, de 31 de marzo, de ordenación e integración de la Formación Profesional. (2022). *Boletín Oficial del Estado*, 78, de 1 de abril de 2022, 43546–43625.
- Real Decreto 659/2023, de 18 de julio, por el que se desarrolla la ordenación del Sistema de Formación Profesional. (2023). *Boletín Oficial del Estado*, 174, de 22 de julio de 2023, 106265–106464.
- Sommerville, I. (2016). *Ingeniería de software*. Pearson Educación.