OBJETIVOS DE LA ASIGNATURA Y SU EVALUACIÓN

1. Objetivo de la asignatura.

Todos los objetivos de la asignatura y las correspondientes competencias como resultado de su aprendizaje, que se indican en la Guía de la Asignatura, se refieren al **Diseño de Software**. Para entenderlos y situar el aprendizaje en este contexto, es posible que convenga definir el término.

Sin entrar en la absurda controversia de si es un arte o una disciplina rigurosa (por el hecho de compartir *la creatividad*), lo cierto es que el significado comúnmente aceptado para la acción de "diseñar" es el de «*el proceso para obtener la representación de una solución a un problema o que cumpla con unos objetivos planteados*»; de la misma forma que el significado del sustantivo "diseño" es el del resultado de ese proceso: «*la representación que especifica esa solución*». En el caso de la asignatura de Diseño de Software, el objetivo del aprendizaje no puede ser otro que la adquisición de las capacidades necesarias para elaborar representaciones que especifiquen un código cuyo comportamiento solucione el problema indicado o satisfaga los objetivos planteados (funcionales y cualitativos). Así mismo, la evaluación de la asignatura debe alinearse, necesariamente, con su objetivo fundamental para el aprendizaje: elaborar un diseño. Sin embargo, falta un ingrediente primordial en ese objetivo: el diseño, la solución obtenida, no sólo debe cumplir con unos requisitos funcionales, sino que debe estar provista de unos atributos cualitativos. En las asignaturas de programación precedentes, la exigencia se limitaba a que la solución fuera funcionalmente válida; es decir, que el diseño (o la implementación) funcionara según los requisitos funcionales pedidos. Al definir qué es diseño, dentro del ciclo de vida del software y en la asignatura de Introducción a la Ingeniería del Software, se establecieron esos criterios cualitativos de la organización del código o del diseño de la solución obtenida: la **independencia funcional**, su **comprensibilidad** y su **adaptabilidad**.

En este punto es imprescindible resaltar el término *especificación*. **Especificar** significa definir, con amplitud y detalle, todos los elementos, funciones, relaciones, características, procedimientos de construcción, manejo o funcionamiento de algo. La especificación debe ser opuesta a la ambigüedad y a la falta de explicación eficaz o de significado. En el ámbito técnico de esta asignatura, el término se refiere a que **un diseño es la especificación de la solución que se persigue**. Es decir: un diseño define, con amplitud y detalle, todos los componentes, características, funcionamiento y comportamiento del código que soluciona el problema planteado o que cumple con unos objetivos funcionales y cualitativos determinados. Evidentemente, la capacidad para especificar depende tanto

del grado formal del sistema de representación que se utilice (el UML para representar los modelos del diseño), como de la naturaleza de lo que se está representando (diseño arquitectónico frente al diseño detallado, descripción de atributos cualitativos frente a la de la funcionalidad y del comportamiento *local* de una función).

La asignatura se refiere al diseño de software y no se limita a describir sólo el comportamiento de una aplicación en relación con las interacciones que se producen con el entorno en el que funciona, ni exclusivamente el de cada uno de sus módulos y las colaboraciones que se producen entre ellos y con el exterior a la aplicación (diseño arquitectónico), como tampoco se limita a hacerlo del código concreto que realiza un caso de uso o una operación (diseño detallado). Aunque se puedan distinguir entre estas perspectivas o estratos en la representación del diseño, cada una de esas especificaciones está incompleta sin las otras. Por otro lado, el alcance de la capacidad para la especificación (su definición, su falta de ambigüedad) es siempre 'de abajo a arriba' en cuanto a la granularidad del detalle: el diseño detallado es una representación cuasi-univoca del código (la propuesta de solución definitiva) y, por agregación y composición, permitiría llegar la vista del diseño arquitectónico; mientras que, al revés, falta definición y no es obvio el camino para obtener una descripción completa.

Quede bien entendido que "diseño de software" es un término integrador que se refiere a todas las perspectivas de la organización y la descripción (especificación) de un único objeto: la totalidad del código que se construye. Cada uno de esos puntos de vista es complementario con los demás. Es decir, ninguna de las perspectivas define, por separado, el software que se está elaborando.

Esas perspectivas se pueden catalogar de forma horizontal, o por capas, atendiendo a cuál es el interés sobre el que actúa la parte del software que se está describiendo:

- Diseño de la capa de Presentación, Interfaz de Usuario y, en general, de la interacción con los actores que manejan la aplicación o esa parte del software. En algunos casos se puede identificar con la "Vista" en el estilo MVC (Modelo-Vista-Controlador).

- Diseño de la capa de la lógica del negocio. Por "negocio" o "dominio del negocio" se entiende la funcionalidad esencial que realiza la aplicación o la porción del software que se desarrolla. Obviamente, la implementación de las otras 2 capas está supeditada a esta funcionalidad. A pesar de este vínculo, es imprescindible garantizar la **independencia funcional** (acoplamiento débil y cohesión alta), la **comprensibilidad** (legibilidad) y la **adaptabilidad** (flexibilidad, modificabilidad y facilidad de mantenimiento) de esta capa con las demás, así como entre los componentes de cada capa.

- Diseño de la capa de servicios técnicos. Esta capa se refiere al software que gestiona todas las interacciones de la capa de negocio con los actores y subsistemas de apoyo, externos a este software. Se incluyen los servicios de acceso a los datos que utiliza la aplicación que, como en el caso de los otros subsistemas, se localizan ('*almacenan*') en un sistema externo a ella, y es completamente ajena a la responsabilidad de organizarlos y mantenerlos ahí (en el almacén o sistema externo). Precisamente esta capa implementa cómo se intercambian los servicios y los datos entre la capa de negocio y la fachada del sistema de apoyo externo.

Simultáneamente, las perspectivas del diseño se pueden clasificar transversalmente a las anteriores (cada vista se aplica en todas las capas), según la granularidad de lo que están describiendo:

- Diseño arquitectónico. Especifica los componentes o módulos en los que se particiona el software, el comportamiento de cada uno y de qué manera se implementa éste, tanto mediante la colaboración e interacción entre los módulos como mediante la información que se intercambia. Se puede estratificar jerárquicamente según la granularidad de la descripción:

    a. Contexto: descripción de la funcionalidad que realiza la aplicación y de su comportamiento en el entorno en el que funciona.

    b. Descomposición modular: el diseño arquitectónico descrito anteriormente. Corresponde al primer nivel en la descomposición, de la aplicación (o el subsistema software) que se elabora, en la totalidad de sus componentes o módulos. Conceptualmente se podría incluir en este nivel un módulo con los elementos del denominado 'control del flujo de trabajo de la aplicación' (*nivel de aplicación*), cuya función es la de coordinar la colaboración entre los otros módulos que justifica el comportamiento global del resto de los módulos. Por este motivo, si se considera como un componente arquitectónico, es el menos cohesionado y el más fuertemente acoplado a los demás.

    c. En algunos casos, algún módulo de la descomposición anterior puede constituir un subsistema y, a su vez, estar compuesto de otros submódulos. En este nivel, la descripción es idéntica a la definición que se ha dado para la especificación del diseño arquitectónico: el comportamiento de cada submódulo y cómo se justifica en función de la colaboración con los demás. La extrapolación de esta descomposición, hasta el extremo, podría llevar a la descripción del funcionamiento de cada elemento del código o al nivel de la sentencia, pero no es aplicable porque ese ámbito se corresponde con el diseño detallado.

En definitiva, el alcance de la descripción en el diseño arquitectónico se limita al comportamiento, tanto global de la aplicación como de cada módulo, justificado por la colaboración entre ellos. Aunque provee indicios de las cualidades arquitectónicas ('independencia funcional', 'comprensibilidad' y 'adaptabilidad') en ningún caso garantiza que la implementación del contenido de cada módulo las vaya a conseguir, ni que se alcancen los objetivos funcionales deseados.

- Diseño detallado: es la especificación técnica de los elementos del código que constituyen el componente arquitectónico de granularidad más fina. Esta especificación enumera los **elementos de código** que componen un módulo y describe la estructura de cada uno, su comportamiento y de qué manera se implementa éste, tanto mediante la colaboración e interacción con otros elementos como mediante los datos que se intercambian. En cuanto a la granularidad a la que se refiere la descripción, es justo el nivel anterior a la sentencia de código. Es decir, se refiere a las unidades mínimas de código con significación funcional que, en el caso de la OO, son las clases y el funcionamiento de sus instancias. Por lo tanto, **la especificación del diseño detallado** es cuasi-univoca con el código y, desde luego, **no puede dejar ambigüedad sobre el comportamiento del código, su construcción, si alcanza los objetivos funcionales planteados o si cumple con los atributos cualitativos** de 'independencia funcional' (acoplamiento débil y cohesión alta), 'comprensibilidad' y la 'adaptabilidad' (flexibilidad, modificabilidad y facilidad de mantenimiento).

  Aunque el objetivo de la descripción en el diseño detallado es el contenido de los módulos (los componentes de la arquitectura) de granularidad más fina, esto no implica que la elaboración de este diseño se realice módulo a módulo: en el desarrollo 'por casos de uso', los elementos de código que intervienen en su funcionalidad pueden provenir de distintos módulos arquitectónicos. De hecho, es frecuente que el controlador y la clase '*objetivo*' del caso de uso estén en un módulo específico, las estructuras de datos (clases) que se usan extensamente en la aplicación, en otro, mientras que otras de esas estructuras, quizás más especializadas, en otro. Por este motivo, en el desarrollo por **casos de uso** no tiene sentido utilizar la descripción del diseño arquitectónico, se utiliza la especificación detallada de los elementos de código que participan en el funcionamiento del CU; sin menoscabo de que esta forma de describir cómo colaboran determinados elementos de distintos módulos, requiere la especificación del diseño arquitectónico para justificarla. En definitiva, el diseño detallado se apoya y necesita la organización en módulos del diseño arquitectónico, mientras que éste se implementa mediante el diseño detallado.

No hay ninguna correlación en el uso de un tipo de perspectiva u otra (por capas o por granularidad). Si se utiliza la perspectiva en capas, cada una tiene su partición en módulos (diseño arquitectónico en la capa) y esos módulos pueden colaborar entre sí o con los de la capa adyacente. Como en todo el sistema, es imprescindible mantener la **independencia funcional** (acoplamiento débil y cohesión alta) **entre capas**.

En cuanto a la descripción del funcionamiento de los elementos del código (diseño detallado), si se hace por casos de uso, es muy probable que algún elemento de una capa colabore o tenga interacción con los de otra capa adyacente. Por ejemplo, el controlador del caso de uso, en la capa de negocio, con la IU en la capa de presentación o con el acceso a los datos almacenados externamente a través de la capa de servicios técnicos. Sin embargo, si se desarrolla el diseño detallado módulo a módulo, es posible que ninguno de los elementos de un módulo, en una capa, tengan relación directa con los de otra capa adyacente a la suya.

En cualquier caso, para conseguir la **independencia funcional** (con cualquier perspectiva del diseño) es imprescindible **excluir el acoplamiento fuerte**, lo que implica **no compartir** el contenido de objetos que intervengan de forma determinante en la funcionalidad: ni objetos aislados (acoplamiento por contenido), ni pertenecientes a un grupo de cuyo contenido participan–comparten otros objetos (acoplamiento común) ni si ese grupo está en otro ámbito de visibilidad (acoplamiento externo).

En conclusión: El **objetivo** de esta asignatura es adquirir la capacidad para **diseñar**, tanto arquitectónicamente como en detalle, **software**.

Se entiende que el **diseño** es un conjunto de actividades para la **elaboración de representaciones que especifican un código** cuyo comportamiento satisface unos requisitos (especialmente los funcionales) **y** que es funcionalmente independiente, comprensible y adaptable.

Especificar es definir, con amplitud y detalle, todos los componentes, características, funcionamiento y comportamiento del código que soluciona el problema planteado o que cumple con unos objetivos (requisitos) funcionales **y** cualitativos de 'independencia funcional' (acoplamiento débil y cohesión alta), 'comprensibilidad' y 'adaptabilidad' (flexibilidad, modificabilidad y facilidad de mantenimiento).

Ni esta asignatura, **ni ninguna** de las indicaciones de su libro de referencia, se refieren a cómo elaborar el software cuyo comportamiento cumpla con unos requisitos funcionales dados (programación, lógica, algoritmia, estructuras de datos, etc.), **sino a cómo hacerlo para que**, sobre la garantía de que los cumple, **el resultado tenga 'independencia funcional'** (acoplamiento débil y cohesión alta), **'comprensibilidad' y 'adaptabilidad'** (flexibilidad, modificabilidad y facilidad de mantenimiento). El malentendido puede deberse a que los procedimientos para conseguirlo se aplican desde el inicio del desarrollo del software, a la vez que se construye el código para que funcione según los requisitos funcionales.

2. Evaluación.

Una vez concretado el objetivo principal de la asignatura y el ámbito del aprendizaje de sus contenidos, es obvio que la extensión de la variabilidad de los sistemas software y de los recursos que se requieren para especificar completamente una solución software, hacen inviable su evaluación exhaustiva. Por ello, se establecen unas reducciones muy notables en el objeto de la evaluación, sin menoscabo de su función para verificar la adquisición de las competencias de la asignatura.

1. **La evaluación se limita a un caso de uso sencillo**, cuya acción fundamental consiste en unas pocas operaciones simples y conocidas. La primera condición para obtener la solución de un problema es definir con claridad cuál es el problema. Aunque se podría definir con precisión el comportamiento deseado globalmente para una aplicación sencilla (o parte de ella), la cantidad de operaciones que se incluyen en su implementación hacen imposible su descripción detallada (sin banalidades ni ambigüedades) en la evaluación. Por otro lado, la descripción de la solución desde la perspectiva de la definición de los módulos que la componen y de las colaboraciones que se producen entre ellos y con el entorno de la aplicación (diseño arquitectónico) sí sería apropiada, **pero incompleta**. Esto se debe a que esta representación no describe cómo se implementa cada función, por lo que no hay ninguna garantía de que la solución propuesta cumpla con los requisitos funcionales. No obstante, el diseño arquitectónico sí facilita la percepción de los atributos cualitativos globales "independencia funcional", "comprensibilidad" o "adaptabilidad".

2. Por el motivo anterior, la solución que se busca en la evaluación corresponde al **diseño detallado**: **la especificación extensa, precisa y detallada del código que realiza las operaciones** que componen el caso de uso.

Como se ha explicado en el punto 1, la elaboración del diseño detallado necesita el sustento del diseño arquitectónico o, al menos, un planteamiento general de la descomposición modular. Aunque la elaboración del diseño arquitectónico no sea un objetivo de la evaluación, **es imprescindible tener en cuenta la visibilidad de los objetos** que intervienen en el funcionamiento especificado por el diseño detallado y, por consiguiente, el módulo del que proceden, cómo están implementando la colaboración entre los módulos de la arquitectura y qué acoplamiento se origina en ella (acoplamiento que necesariamente se refleja en la descripción del diseño detallado).

Sin contradecir lo último, en el desarrollo por casos de uso es imprescindible situar su funcionalidad con relación a la de los otros CU en la aplicación, los objetos que intervienen en el funcionamiento, su procedencia y visibilidad y, lo más importante, qué operaciones se realizan en la aplicación para llegar a la situación de inicio del caso de uso. Es decir, hay que tener en cuenta el funcionamiento del flujo de trabajo global en la aplicación; lo que se denomina 'nivel de aplicación'. Aunque en la evaluación no se exige el diseño detallado del 'nivel de aplicación', **es imprescindible tenerlo en cuenta para diseñar, en detalle, el funcionamiento del caso de uso**.

3. Para limitar aún más la extensión de la solución, **el diseño detallado se restringe al código de la capa del dominio del negocio**. Es decir, se excluye la especificación detallada del código correspondiente a las operaciones, pertenecientes al caso de uso, que transcurren en la capa de presentación o en la capa de servicios técnicos. En concreto, de los mecanismos de adaptación entre el código de la capa del negocio, perfectamente desacoplado con la capa de IU, la cual permite la interacción con el actor que maneja el caso de uso. También de los mecanismos de adaptación entre el código de la capa del negocio, perfectamente desacoplado con el acceso a los datos, o cualquier otro servicio de la capa de servicios técnicos, los cuales permiten el intercambio de datos o servicios con el almacenamiento externo o cualquier otro sistema de apoyo externo.

   Sin embargo, al igual que en los planteamientos anteriores, la exención de elaborar el diseño detallado de esos mecanismos no exime de su conocimiento profundo, cómo se implementan, qué estrategia de patrones de diseño se utiliza para hacerlo y qué consecuencias tienen para el funcionamiento del resto de elementos funcionales del caso de uso y para las cualidades exigidas en todo el diseño (independencia funcional, comprensibilidad y adaptabilidad).

En conclusión: Como no puede ser de otra manera, las pruebas de evaluación son el medio para valorar, objetiva y eficazmente, la adquisición, por parte del estudiante, de los resultados del aprendizaje en la asignatura. Es decir, valoran **su capacidad para diseñar software** o, lo que es lo mismo, para elaborar la representación de una solución que, formalmente (en UML), identifique sin ambigüedades un código OO y cumpla, simultáneamente:

a. Su comportamiento **satisface los requisitos funcionales** establecidos para el problema planteado.

b. La solución **esté dotada** de **'independencia funcional'** (acoplamiento débil y cohesión alta), **'comprensibilidad'** y **'adaptabilidad'** (flexibilidad, modificabilidad y facilidad de mantenimiento). Como las 2 últimas propiedades dependen fuertemente de la primera, fundamentalmente **se verifica el tipo de acoplamiento** de la solución.

Para que las pruebas sean totalmente compatibles con los objetivos que se evalúan y su realización sea viable en el período máximo de 120 minutos, se establecen las máximas simplificaciones posibles para su elaboración:

1. Los requisitos funcionales que debe satisfacer la solución se limitan a unas pocas operaciones, simples y conocidas, **de un único caso de uso sencillo**. No obstante, el comportamiento de la solución, para ese caso de uso, debe ser compatible y coherente con la funcionalidad general de la aplicación en la que se inscribe.

2. **El desarrollo de la solución**, su cumplimiento de los requisitos funcionales y de la condición de independencia funcional, restringido al caso de uso planteado, **se limita a la capa del dominio del negocio**; excluyendo la capa de presentación (IU) y la de servicios técnicos (servicio de acceso a datos, de acceso a los servicios de otros sistemas de apoyo externos y otros servicios técnicos y de nivel bajo que dan soporte a la aplicación). A pesar de esta reducción, el comportamiento de la solución debe ser compatible, en esta capa del dominio del negocio, con el alcance funcional de las otras 2 capas (lo que debería hacer la solución en ellas), en las que dicha solución no se implementa.

3. Al ser la solución un diseño limitado a un caso de uso, **también se excluye de su elaboración el diseño arquitectónico**. Por consiguiente, la solución pedida **se reduce a un diseño detallado** restringido, únicamente, al funcionamiento de un caso de uso en la capa del dominio del negocio. Por lo que se ha justificado anteriormente, la elaboración del diseño detallado requiere tener en cuenta la descomposición modular, cómo es la colaboración entre los módulos, la interacción entre las capas, el comportamiento en el 'nivel de aplicación' y el control global del flujo de trabajo; aspectos muy importantes de la asignatura que **no** quedan excluidos de la evaluación.

4. La representación de la solución pedida, del diseño detallado, se realiza con un lenguaje de modelado, UML, con potencia y capacidades sobradas para hacerlo. Para representar el comportamiento del código OO que especifica el diseño detallado, su funcionamiento preciso y cómo interviene en él cada clase y cada instancia de la solución, el mayor alcance de este lenguaje se obtiene con los diagramas de interacción: el diagrama de secuencia, detallado y completo, del caso de uso o la colección de diagramas de colaboración para todas sus operaciones. Por no ser un objetivo de esta asignatura, la última simplificación se refleja en que no hay un control estricto con el uso del lenguaje UML y su sintaxis. Sin embargo, sí se exige que esta representación del diseño detallado especifique el código OO de la solución y su comportamiento (imprescindible para poder evaluar), por lo que se es **notablemente riguroso en los casos de ambigüedades, omisiones o inconsistencias flagrantes** que incapaciten, a esa parte de la representación, para especificar un comportamiento verificable en el código.