CRITERIOS DE CORRECCIÓN DE LAS PRUEBAS DE EVALUACIÓN

CONSIDERACIONES GENERALES

En este tipo de ejercicios no hay una solución única y cerrada, las respuestas deben considerarse en el contexto de **lo útiles que sean**. En los documentos de guía de la asignatura y, en concreto, en el de "Recomendaciones para las Evaluaciones", se establece que el objetivo del diseño es la obtención de un código que se comporte según lo deseado, por tanto, **el objetivo principal de la asignatura es mejorar la destreza y eficacia del estudiante en la codificación del software** de las aplicaciones, incorporando al diseño nuevos requisitos, cualitativos y no estrictamente funcionales, como la flexibilidad del código o su mantenibilidad.

La elaboración del análisis y diseño debería recoger la mayoría de los aspectos '*principales y obvios*' e intentar transmitir, con claridad, qué es lo importante que se debe saber acerca del problema y sus soluciones. Para paliar la ambigüedad de la imagen de que «*las soluciones son abiertas*» o «*aquí vale todo, según se interprete*» se incorpora a la asignatura una buena dosis de *formalismo* (rigor, en realidad), mediante el lenguaje UML y, sobre todo, mediante **la verificación** de que cualquier especificación elaborada sea codificable y de que el comportamiento de ese código cumpla con los requisitos planteados. Es decir: aunque no se sea estricto en el uso de la sintaxis de UML, lo que se representa en un diagrama sólo se puede interpretar, en el contexto del problema, de una manera; y lo que se evalúa es si la unión de esos diagramas es coherente y especifica, de forma unívoca, un código cuyo comportamiento es el pedido.

El párrafo anterior se refiere a una situación de mínimos, condición necesaria para aprobar la evaluación. Es decir: que las respuestas elaboren una especificación de diseño, no ambigua, que signifique, de forma cuasi-unívoca, un código en Java cuyo comportamiento sea el pedido por el enunciado para el caso de uso. Sin embargo, el objetivo fundamental y diferenciador de la asignatura es que ese diseño, además de ser realizable y coherente con el comportamiento pedido, debe aportar flexibilidad funcional al código y facilidad para su mantenibilidad. Por ello, esta característica también se exige en todas las evaluaciones. La tesis que se mantiene en la asignatura es que esa flexibilidad en el código se obtiene a través de la aplicación adecuada de los principios GRASP en su diseño. En conclusión:

- La prueba está sólo superada (**máxima calificación: 6**) si, y sólo si, a través de las respuestas a las preguntas del Modelo de Dominio, el DS, los Diagramas de Colaboración de las Operaciones y el DCD, se obtiene una especificación de diseño, no ambigua, que signifique, de forma cuasi-

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

swdesign@issi.uned.es

unívoca, un código en Java (u otro formal) cuyo comportamiento sea el pedido por el enunciado para el caso de uso.
- Si, a partir de las condiciones anteriores, se utiliza un manejo adecuado de los principios GRASP (especialmente los 5 primeros), de manera que el diseño resultante también añada esa característica de flexibilidad al código, la calificación podrá ser **superior a 6**.
- Opcionalmente, se podría calificar con **excelencia** los ejercicios en los que, además de lo anterior, se demostrara la utilización acertada de los patrones GoF de diseño, especialmente los que se derivan de los 4 últimos principios GRASP.

Hay que tener en cuenta que los Casos de Uso no tienen ninguna relación con la orientación a objetos por lo que, de su elaboración, no se desprende ninguna medida de las capacidades del estudiante relacionadas con el objetivo central de esta asignatura. Por el contrario, las habilidades mostradas en la asignación de responsabilidades y en el diseño de las colaboraciones, son las de mayor importancia porque:

- Son ineludibles. Que quede claro, una vez más, que lo que se califica en las respuestas del estudiante son las especificaciones que ha desarrollado, cómo las expresa y su capacidad, unívoca, para construir el código que se comporte según se pide en el enunciado.
- Influyen fuertemente en la calidad, escalabilidad, flexibilidad y mantenibilidad de los sistemas software.

RESPUESTAS

Véase el documento "Recomendaciones para las Evaluaciones", donde se explica qué se pretende en la respuesta de cada pregunta.

SECCIÓN 1. EVALUACIÓN DE LOS *CASOS DE USO*

1. (0'5 PUNTOS) IDENTIFICACIÓN DE CASOS DE USO PRIMARIOS, SUS ACTORES PRINCIPALES Y DE APOYO Y UBICACIÓN DEL SISTEMA RESPECTO A SU ENTORNO DE FUNCIONAMIENTO.

Se debería obtener algo similar a los servicios que provee la aplicación a los usuarios genéricos, o a alguno específico, y que se indican en el enunciado. Un aspecto muy importante es saber diferenciar los casos de uso primarios (operación o servicio principal del subsistema software que se está analizando, normalmente se corresponde con una **funcionalidad esencial** del sistema o un **objetivo fundamenta**l en el uso del sistema por parte de un actor principal) respecto de las operaciones, funciones y casos de uso secundarios.

ERRORES FRECUENTES

- Un error común es representar un paso, una secuencia de un caso de uso o una operación, que no es objetivo principal de uso para el actor, como un caso de uso separado. Por ejemplo, no son aceptables como casos de uso primarios Imprimir billetes, Registrar datos de pago o Autentificarse. Similar a lo anterior, es presentar como casos de uso primarios operaciones incluidas, <<include>>, o que amplían la funcionalidad, <<extends>> (con frecuencia, opcionalmente), de un caso de uso primario. O viceversa. Ya que no se pide, si se indican casos de uso secundarios u operaciones de apoyo, al menos, que no se confundan <<extends>> con <<include>> (ver "UML use case EXTEND vs INCLUDE relationships.pdf").
- Pasar por alto actores de apoyo o subsistemas que no deberían estar embebidos en el software de estudio y, sin embargo, juegan un papel importante en los casos de uso primarios que se exponen.
- No representar alguna interacción importante entre los actores, principales o de apoyo, y alguno de los casos de uso. O establecer, erróneamente, un vínculo entre un caso de uso y un actor basándose en una interacción que no existe.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- No saber representar un diagrama de casos de uso del sistema estudiado.
- No encontrar un actor principal adecuado, generalmente obvio según se describe en el enunciado.
- Excepcionalmente, no encontrar ningún sistema o actor de apoyo por considerarlo incluido en el software de los casos de uso primarios. Es decir, no poder determinar los límites del sistema de estudio.

2. (1 PUNTO) LA ESCRITURA DEL CASO DE USO

En esta pregunta se plantea un escenario para un caso de uso. Se indica su situación en relación al sistema software estudiado, su inicio, su término y en qué consiste, con la mayor precisión posible. **Este caso de uso** (y ningún otro) es el objetivo de todas las preguntas, de aquí en adelante.

ERRORES FRECUENTES

- No exponer con claridad la contraposición entre las *Acciones del Actor* y las *Respuestas del Sistema*. La exposición en el formato en dos columnas ayuda a hacerlo correctamente.
- No identificar, correctamente, el inicio y la situación final de éxito en el caso de uso. El primer paso debería comenzar con el inicio canónico "*El caso de uso comienza cuando el <Actor>…*".
- Alterar el orden temporal de alguna parte de la secuencia de acciones, llevando a situaciones difíciles de entender o antinaturales en el desarrollo expuesto para el caso de uso.

- Puesto que se especifica '*estilo esencial*', se deben evitar detalles relacionados con los diversos medios y formatos de entrada/salida (como *lectores de tarjetas*, *apretar botón*, etc.). De igual forma, el relato de las operaciones realizadas por el sistema, tampoco debe contener referencias a cómo lo hacen o a sus detalles técnicos; más allá de la secuencia de pasos u operaciones para hacerlo.
- Incluir alternativas, además de la típica sucesión de acontecimientos con finalización satisfactoria y sin problemas, en el caso de uso. El flujo principal es un aspecto muy importante para la calificación, puesto que es la idea central de los casos de uso.
- En cuanto a los flujos alternativos, la ausencia de los que corresponden a situaciones muy frecuentes puede reflejar un análisis muy pobre. En cualquier caso, son líneas completas y, por lo tanto, se deben enumerar todos los pasos, desde la bifurcación y su condición, hasta su término o su reconexión con la línea principal de éxito.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- Interpretar un caso de uso que **no es** el enunciado en la pregunta y que conlleva un comportamiento sustancialmente distinto al pedido.
- Alterar el orden temporal de alguna parte de la secuencia de acciones, de forma que el desarrollo del caso de uso sea notablemente distinto al expuesto en el enunciado.

SECCIÓN 2. EVALUACIÓN DEL *MODELADO CONCEPTUAL*

3. (2 PUNTOS) LA ELABORACIÓN DEL MODELO DE DOMINIO

El modelo de dominio **debe reflejar el funcionamiento** *operativo* del caso de uso, no del software. Para ello, se deben construir unas clases de objetos (objetos *conceptuales*) que puedan realizar las acciones que se han descrito en el flujo principal de éxito elaborado, en la pregunta anterior, para este caso de uso. Es decir:

1. Que puedan recibir los estímulos generados por los actores.
2. Que puedan realizar las operaciones indicadas en la escritura del caso de uso.
3. Que, en el caso de que requiera algún dato o información adicional para realizar alguna acción correspondiente al rol funcional o de comportamiento que se le ha asignado, un objeto pueda obtenerlo de otro o a través de un actor de apoyo.

ERRORES FRECUENTES

- Que falte algún componente, alguna clase conceptual o alguna relación entre ellos, necesarios para explicar alguna parte del funcionamiento descrito para el caso de uso.
- Que no se entienda el papel que cumple (el rol funcional), en coherencia con lo descrito en la pregunta anterior, de alguno de los objetos.
- No indicar la información (atributos) que maneja cada objeto y que, precisamente, explica para qué sirven, cuál es su papel en el modelo.
- Identificar un objeto según una función que realice (verbo), y no en virtud de la entidad de información sobre la que realizan las operaciones correspondientes al rol funcional que se les asigna; que sería lo correcto. Los nombres de las relaciones entre objetos **tampoco** son los verbos que se identificarán con las invocaciones a los servicios solicitados en el objeto relacionado, si no que se refieren a la interacción con el actor, a la ubicación de la información necesaria para efectuar la operación o el lugar donde se realiza. De igual forma, **tampoco** son los argumentos transmitidos en esa supuesta invocación de un servicio (*llamada a función*), **ni** los datos o la información que comparten los objetos.
- No indicar el sentido de las relaciones o su cardinalidad.
- Romper la privacidad de la información que manejan los objetos mediante agregaciones indiscriminadas.
- No aportar una solución a algún mecanismo que se prevé necesario: búsqueda de información (*Catálogo*), registro o captura de información relevante (*Registro*, identificación de *objetos componentes*), etc.
- Desde la justificación de que, a excepción de la información o datos específicos que aporte el actor principal, toda la información que maneja el caso de uso se obtiene de algún almacén, o sistema de apoyo externo, a través de mecanismos software (conexión y consulta a elementos externos (*adaptadores* y *polimorfismo*), persistencia, etc.; justificación desarrollada en el documento "Abstracción de los servicios de acceso a datos"), **no se deberían incluir los objetos** (software) **que implementarán dichos mecanismos**. Por el contrario, esa información, necesaria para explicar la lógica del funcionamiento del caso de uso (la lógica del uso), **estará disponible** en el modelo y **la labor consistirá en organizarla en objetos de manera que posibilite su utilización**. Por consiguiente, también es erróneo incluir un objeto software: el que sería el artífice para realizar alguna maniobra que, no estando prevista en ninguna de las operaciones que se desprenden de la escritura del caso de uso, será necesaria para que el software la lleve a cabo.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- Que no se incluyan las clases y las relaciones necesarias, se llamen como se llamen, para poder ver con claridad (no *imaginar* ni *interpretar*) cómo funciona el sistema bajo estudio, en coherencia con la respuesta a la pregunta 2. En general, no se puede aceptar que el diagrama no explique, completa y coherentemente, el caso de uso pedido.
- Incluir un elemento, que debería ser un subsistema o actor de apoyo externo, como componente activo del modelo. Por ejemplo, las bases de datos o un SGBD.
- Representar un DER, un DTE, DFD, DCD o cualquier otro diagrama que no sea el pedido.

SECCIÓN 3. EVALUACIÓN DE LOS ***EVENTOS DEL CASO DE USO*** (INICIO DE ***ASIGNACIÓN DE RESPONSABILIDADES***)

4. (**3 PUNTOS**) EL DIAGRAMA DE SECUENCIA Y LA ESCRITURA DE LOS CONTRATOS DE LAS OPERACIONES

4.1. (**2 PUNTOS**) EL DIAGRAMA DE SECUENCIA

En esta pregunta no sólo se evalúan los eventos que recibe el software del caso de uso si no que se empieza a representar la **especificación del diseño software**. Por ello, hay que tomar decisiones en la asignación de responsabilidades a los objetos. Así que, en esta pregunta, **no se pone la interacción entre el actor principal y el sistema como un único bloque—objeto que representa software del caso de uso**; si no que **hay que desglosar esa interacción, entre los distintos objetos del modelo de dominio**: cuál recibe los eventos (generalmente el controlador), cómo se desencadenan las acciones hacia los otros objetos del sistema, cómo se recogen sus respuestas y cómo se presentan los resultados o las indicaciones al actor para que genere un nuevo evento y se complete el caso de uso.

Aun realizándolo con el desglose de los objetos constituyentes del sistema, este DS se puede hacer de dos maneras: simplificado o detallado. En la forma simplificada se representan los eventos que dan lugar al funcionamiento descrito para el sistema, qué objeto los recoge y los gestiona (controlador), qué acciones principales desencadena este ante la llegada de cada evento y qué otros objetos intervienen en ellas. El detalle de los eventos y de las acciones que se desencadenan debe ser el necesario para reflejar todas las operaciones que componen el caso de uso y para poder escribir los contratos de las operaciones elegidas (porque contiene todos los elementos de cada contrato); todo ello en coherencia con el comportamiento que se ha establecido anteriormente. Todos los detalles que se eviten en esta modalidad simplificada (lo justo para escribir los contratos de las operaciones adecuadamente) habrá que desarrollarlo, exhaustivamente, para hacer los diagramas de colaboración.

ERRORES FRECUENTES

- Si, a pesar de indicarlo explícitamente en el enunciado, el '*bloque*' del sistema no se desglosa en los objetos software que definen su comportamiento, **la puntuación de esta pregunta descenderá a 0'5 puntos** y sólo se evaluará la adecuación de los eventos considerados.
- Puede haber una gran variabilidad en los nombres de los eventos, pero deben indicar con claridad las operaciones y su naturaleza.
- Los eventos se traducen en una invocación a un método del controlador. Por tanto, el nombre de un evento debería empezar con un verbo.
- Que varios objetos asuman el rol de controlador y atiendan, simultánea e indiscriminadamente, los diferentes eventos de un mismo actor.
- Incluir un evento que se produce en una entrada de nivel medio - bajo (por ejemplo, *introducirDNIViajero(dniViajero)*). Es decir, el detalle de una entrada de información que, debido a su granularidad, es irrelevante para la operación que se está implementando.
- No utilizar los argumentos adecuados. Tanto los eventos como las acciones que se desencadenan en los objetos, son invocaciones a funciones. No utilizar argumentos o que sean incorrectos o incoherentes con los atributos de los objetos, tal como se ha descrito en el modelo de dominio, tergiversa notablemente su validez para especificar el código.
- Utilizar eventos no coherentes con el modelo de dominio definido anteriormente.
- Que algún objeto envíe una acción, en forma de código o invocación a una función, a un actor humano. Estos actores **no tienen capacidad para aceptar código**. La información que envíe el sistema software, se hace a través de la IU para que sea formateada y representada de manera que pueda recibirla el actor. En este tipo de **respuesta**, se indica únicamente la naturaleza de la información, no qué hay que hacer con ella, cómo manejarla ni, mucho menos, una estructura de datos o comando.
- Utilizar objetos no coherentes con el modelo de dominio definido anteriormente, sin estar justificado en una necesidad del funcionamiento del propio software (objetos *Factory*).
- Que una acción se genere desde un objeto que no tiene la capacidad de hacerlo, porque no disponga de la información necesaria para hacerlo o porque no pueda manejar la reacción a dicha acción.
- Que una acción se produzca en una situación de la línea temporal inadecuada o incongruente con las acciones anteriores o con la existencia de otros objetos. Un ejemplo particular es el de la creación de una instancia. Otro ejemplo es que se produzca la acción sin que exista un estímulo para ello.
- No representar un objeto en el lugar temporal de su creación.

- Los mensajes a un objeto múltiple (colecciones de objetos) deberían dirigirse al propio objeto y no de forma indeterminada a los miembros de la colección (ver, en el libro o en las soluciones propuestas para las PEC anteriores, el mecanismo para agregar una instancia al multiobjeto o para modificarla).
- En el documento Recomendaciones para las Evaluaciones, se ha justificado la pertenencia de esta pregunta a las actividades del diseño. De esta forma, buena parte de los defectos descritos anteriormente se identifican con un uso nulo, o inadecuado, de los principios GRASP. La falta de identificación y asignación de los roles de Controlador, Experto o Creador a objetos concretos (actividad que se debería iniciar ya en el modelo de dominio) y la de su articulación con el funcionamiento pedido, si bien no es preceptivo que se incluya en las respuestas de esta pregunta, en casos que no son los peores, va a producir otros defectos importantes en el resultado final del diseño. Me refiero a un acoplamiento alto y a la rigidez funcional, o para el mantenimiento, que este tipo de diseños produce en el código. Aunque se consiga implementar el comportamiento pedido para el caso de uso (condición mínima para aprobar la evaluación), esa característica de flexibilidad, en el resultado de lo que se ha diseñado, es el objetivo fundamental y diferenciador de esta asignatura; exigida, por tanto, en todas las evaluaciones. Si no se aplican, ya aquí, los principios GRASP, se van a producir defectos en el diseño que, aunque pueden no afectar a la puntuación mínima en esta pregunta, sí puede hacerlo, notablemente, en las siguientes.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- En general, no se puede aceptar que el diagrama no explique, completa y coherentemente, el funcionamiento del software para el caso de uso pedido o que dicho funcionamiento sea sustancialmente distinto al que expresa el enunciado.
- La situación anterior también ocurre si el '*bloque*' del sistema no se desglosa en los objetos software; ya que no se puede saber qué responsabilidad se asigna a cada objeto. En ese caso, tampoco se puede justificar lo que se exprese en la escritura de los contratos de las operaciones ni en los diagramas de colaboración.

4.2. (**1 PUNTO**) LA ESCRITURA DE LOS CONTRATOS DE LAS OPERACIONES

En la evaluación, o el enunciado especifica las dos operaciones de estudio o hay que seleccionarlas de forma que den una cobertura *casi* completa al funcionamiento del caso de uso o, al menos, que cada una tenga una entidad apreciable en cuanto a la importancia del objetivo que la colaboración entre las acciones del software pretende alcanzar, dentro del funcionamiento del caso de uso.

ERRORES FRECUENTES

- Si, a pesar de indicarlo explícitamente en el apartado 4.1) del enunciado, el '*bloque*' del sistema no se desglosa en los objetos software que definen su comportamiento, **la puntuación de esta pregunta descenderá a 0'2 puntos** y sólo se evaluará la adecuación de los eventos considerados, la sintaxis con la que se expresan los contratos y la adecuación al enunciado de la especificación del comportamiento del software que representan.
- Puede haber una gran variabilidad en los nombres de las operaciones, pero deben indicar con claridad las acciones o funciones que realiza y su naturaleza. Además, el nombre de una operación debería empezar con un verbo.
- Seleccionar como operación principal la que está motivada por una entrada de nivel medio - bajo (por ejemplo, *introducirDNIViajero(dniViajero)*).
- Utilizar operaciones no coherentes con el modelo de dominio o que no se reflejen en el DS, definidos anteriormente.
- Utilizar objetos que no existen en el DS o que su comportamiento no sea coherente con el descrito en él.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- En el caso de que no se pueda saber qué responsabilidad se asigna a cada objeto (porque en el apartado 4.1) de esta pregunta no se haya desglosado el DS), tampoco se puede justificar lo que se exprese en la escritura de los contratos de las operaciones. Es decir, la escritura de las operaciones, aunque sintácticamente sea correcta, no aporta nada sustancial respecto al funcionamiento del software o la especificación para construir el código correspondiente; por lo que no cumple su objetivo.
- No expresar las postcondiciones (o responsabilidades) en términos de creación de instancias, creación de asociaciones, asignación de valores, etc. (sintaxis y expresión cuasi-formal). Igualmente, no expresar las precondiciones en esos mismos términos o que las postcondiciones dependan de precondiciones que no están enunciadas explícitamente.
- Utilizar operaciones absurdas, sin sentido o sin ninguna coherencia o sin justificación en lo que se ha reflejado en el modelo de dominio y en el DS.
- En general no se puede aceptar la escritura del contrato de una operación, si lo expresado en ella, especialmente en las postcondiciones, no es una especificación clara de cómo debe funcionar el código, de acuerdo a la responsabilidad que se le ha asignado.

SECCIÓN 4. EVALUACIÓN DE LA *ASIGNACIÓN DE RESPONSABILIDADES* Y *DISEÑO DE COLABORACIONES*

5. (**1 PUNTO**) DIAGRAMA DE COLABORACIÓN DE LA OPERACIÓN A.
6. (**1 PUNTO**) DIAGRAMA DE COLABORACIÓN DE LA OPERACIÓN B.

Los diagramas de colaboración, aunque sean fragmentos por operaciones, expresan exactamente lo mismo que el diagrama de secuencia elaborado en 4.1. Además, estos dos diagramas difieren, ligeramente, en su sintaxis gráfica. Se podrían resaltar las siguientes cuestiones:

- Manejar muchas opciones es algo inherente al diseño y, por ello, debería ser aceptable que haya cierta variabilidad en las soluciones. Lo importante, como en las respuestas anteriores, es que el funcionamiento descrito se atenga al solicitado en el enunciado, que sea coherente con la línea fijada en las respuestas anteriores y que el estudiante haga un uso juicioso y sensato de los principios GRASP.
- Debe dejar bien claro (en su denominación y en las responsabilidades que tiene) cuál es el controlador de la operación.
- Una vez definido el controlador, se deben reflejar con claridad (mediante el paso de mensajes) las colaboraciones con otros objetos, las transacciones, las creaciones y las consultas; de manera que se efectúe la operación tal como se indica en el contrato y de manera coherente al transcurso de acontecimientos de la escritura del Caso de Uso.
- Un estudiante experimentado usaría algún patrón (GoF) para manejar las operaciones del sistema; aunque quizás sea una pretensión ambiciosa para uno principiante. (El uso adecuado de patrones GoF aumenta notablemente la calificación, siempre y cuando –claro está— el desarrollo general y la aplicación de los principios GRASP sean correctos).
- Los mensajes a un objeto múltiple (colecciones de objetos) deberían dirigirse al propio objeto y no de forma indeterminada a los miembros de la colección (ver, en el libro o en las soluciones propuestas para las PEC anteriores, el mecanismo para agregar una instancia al multiobjeto o para modificarla).

ERRORES FRECUENTES

Los mensajes son invocaciones a responsabilidades (métodos) que residen en el objeto de destino. Son errores muy comunes:

- La clase destinataria del mensaje no tiene asignada esa responsabilidad.
- El mensaje no contiene la información necesaria (argumentos) para realizar la acción que invoca. Debido a que la clase receptora no maneja la información que necesita o ésta no se recibe en el mensaje, el objeto no puede llevar a cabo esa responsabilidad.

- Un objeto necesita una información, pero o no se la pide a otro, Experto en esa información, o no aparece como resultado del mensaje (invocación) al Experto.
- Una clase acapara gran cantidad de información, por lo que detenta una sobrecarga de responsabilidades. Transgrede varios principios GRASP importantes. Esto también se puede apreciar en el Modelo de Dominio, con una gran cantidad de relaciones de composición y un número elevado de objetos que son 'componentes' de otro (el '*sobrecargado*').

Además, en general:

- El uso de los principios GRASP es inherente al diseño y es obvio que afecta a las respuestas de todas esas preguntas. Pero también afecta al Modelo de Dominio, en el que se apunta un candidato para ser Controlador o en el que se hace una pseudo-declaración de la privacidad de los objetos (principio de Experto). A pesar de esta repercusión tan amplia, es en esta pregunta donde más se evidencia que las dificultades para construir un diseño correcto o sus defectos, mencionados anteriormente, se deben a una aplicación inadecuada de los principios GRASP.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- Que no se numeren los mensajes según el orden de ejecución y, por tanto, no se pueda determinar la secuencia en la que se producen las acciones. En ese caso, el diagrama no tiene validez para construir el código.
- La secuencia de los mensajes no es coherente con lo descrito en el contrato o con la secuencia de operaciones del Caso de Uso.
- Si, a pesar de que en el apartado 4.1) no se ha desglosado el DS, aún aquí tampoco se puede determinar cómo se comporta cada objeto; o sí, pero el comportamiento resultante que se deduce no es acorde con el que se pide en el enunciado o que la forma de representarlo no permite la construcción del código que se comporte según lo pedido.
- En general no se puede aceptar el diagrama si (aunque se haya desglosado el DS en 4.1) ocurre algo del anterior punto o se da alguna incoherencia, respecto a lo establecido en los diagramas antecesores, que lleve al mismo resultado indeseado. Por ejemplo, cambios injustificados de nombres en los objetos o de los roles o las responsabilidades (mensajes, componentes o atributos) que se les ha asignado, o *aparición* de otros nuevos *de la nada*.

SECCIÓN 5. EVALUACIÓN DE LOS ***DIAGRAMAS DE CLASES*** DE DISEÑO

7. (**1 PUNTO**) DIAGRAMA DE CLASES DE DISEÑO.

Durante la elaboración del DS (punto 4.1) y los diagramas de colaboración (puntos 5 y 6), es **imprescindible modificar el modelo de dominio** con los hallazgos y conclusiones **que afecten a la lógica de la aplicación**; **no así** con los objetos estrictamente software obtenidos (fabricación pura o auxiliares para la *mecánica* de la implementación) ni con las asociaciones encontradas ahí; que sí se deben incluir en este diagrama.

ERRORES FRECUENTES

- Que el diagrama no se centre en el objeto que se pide o sobre el que se realiza la operación fundamental que define al caso de uso.
- Que, aunque el diagrama se centre en el objeto pedido, no recoja las modificaciones que se deberían hecho, posiblemente, en el Modelo de Dominio, como consecuencia de algunas conclusiones obtenidas durante el diseño.
- Un error muy común es que alguna clase contenga una gran cantidad de instancias, de otras clases, como atributos *componente*; en lugar de establecer asociaciones con su navegabilidad. Significa un mal diseño, próximo al monolítico, altamente acoplado, seguramente con clases sobrecargadas y con poca diferenciación en los roles y las responsabilidades que les corresponden. Lo contrario a lo que se busca.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- No se añaden las clases software adicionales que se puedan haber encontrado en el desarrollo del DS, o en los diagramas de colaboración, para las operaciones desarrolladas.
- No se incorporan los métodos, que constituyen las responsabilidades asignadas e implementan la funcionalidad del caso de uso, a cada una de las clases del diagrama. Se deben excluir los *getters* y *setters*.
- No se añaden las asociaciones obtenidas en la escritura de los contratos de las operaciones.
- En general no se puede aceptar un DCD que sea una reproducción, incluso resumida, del Modelo de Dominio, o que no incorpore los hallazgos obtenidos en los pasos anteriores del diseño, o que no contenga la privacidad de cada clase (atributos) y su capacidad para realizar las responsabilidades (métodos y relaciones) que se le han asignado. En definitiva, no es admisible si el diagrama no contiene toda la información necesaria para obtener, directa y unívocamente, un código que realice lo que se pide en este caso de uso.

SECCIÓN 6. EVALUACIÓN DE LA ***TRANSFORMACIÓN DEL DISEÑO EN CÓDIGO***

8. (**0'5 PUNTOS**) CODIFICACIÓN.

La transformación en código que se pide aquí, se refiere al DCD de la pregunta anterior. Todos los métodos significativos se han tenido que escribir, ya, en el DS y los diagramas de colaboración; con sus argumentos, y sus tipos, definidos y adecuados. Bastaría con trasladarlos aquí y darles un formato de definición se clase. Faltaría por decidir qué métodos son los más representativos de las 2 operaciones principales, significantes de la ejecución del caso de uso; de acuerdo con todo lo anterior. No es necesario ser estricto en la sintaxis (se puede usar pseudocódigo) ni tampoco desarrollar el *cuerpo ejecutable* de los métodos.

ERRORES FRECUENTES

- Utilizar nombres de atributos, argumentos o métodos que no aparecen en los diagramas anteriores del diseño, o que están asignados a clases diferentes.
- No definir el tipo de atributos, variables, argumentos o de los valores que devuelven los métodos.
- Puede haber una gran variabilidad en los nombres de las operaciones, pero deben indicar con claridad las acciones o funciones que realiza y su naturaleza. Además, el nombre de una operación debería empezar con un verbo.
- Seleccionar como operación principal la que está motivada por una entrada de nivel medio - bajo (por ejemplo, *introducirDNIViajero(dniViajero)*).
- Utilizar operaciones no coherentes con el modelo de dominio o que no se reflejen en el DS, definidos anteriormente.
- Utilizar objetos que no existen en el DS o que su comportamiento no sea coherente con el descrito en él.

ERRORES INACEPTABLES PARA SUPERAR LA PREGUNTA

- Que no exista una coherencia estricta entre este código y el contenido del DCD. Evidentemente, si no existe respuesta para el DCD, la coherencia debe provenir de las anteriores etapas de diseño y análisis.
- Que el comportamiento de este código, aunque no sea completo ni esté totalmente definido, no se corresponda con el pedido en el enunciado para el caso de uso.