Listas de Chequeo para Validación de SRS y Modelos UML en Metodologías Ágiles

La validación del **Documento de Requisitos de Software (SRS)** y de los **modelos UML** es un paso crucial durante la etapa de diseño de un proyecto de software. A continuación, se presentan listas de chequeo para guiar a aprendices de programación en esta tarea, cubriendo dos escenarios: la revisión del SRS y la revisión de modelos UML (diagramas de casos de uso, clases y actividades). Cada escenario incluye cuatro versiones de la lista, adaptadas a distintas metodologías de desarrollo ágil: **Scrum**, **Extreme Programming (XP)**, **Rational Unified Process (RUP)** y **Rapid Application Development (RAD)**.

Estas metodologías difieren en su manejo de requisitos y documentación: por ejemplo, enfoques ágiles como Scrum y XP priorizan el software funcional sobre la documentación extensiva, mientras que RUP tiende a producir numerosos artefactos formales. Por ello, cada lista de chequeo enfatiza criterios alineados con las prácticas típicas de cada enfoque.

Cada tabla de verificación contiene columnas para:

- **Criterio de evaluación** – el aspecto específico a verificar.
- **Descripción** – explicación clara del criterio en contexto.
- **Cumple (Sí/No)** – espacio para marcar si se cumple el criterio.
- **Observaciones** – espacio para anotar comentarios o hallazgos durante la revisión.

Validación del Documento de Requisitos de Software (SRS)

Scrum

Criterio de evaluación	Descripción	Cumple (Sí/No)	Observaciones
Cobertura de requisitos	El Product Backlog incluye todas las funcionalidades requeridas por el cliente, representadas en historias de usuario o épicas Se verifica que todas las necesidades del cliente estén reflejadas en el backlog como historias de usuario claras, sin omitir requisitos importantes.		
Criterios de aceptación	Cada historia de usuario tiene criterios de aceptación claramente definidos y verificables. Los criterios de aceptación describen cómo comprobar que la historia se implementó correctamente, sirviendo como base de pruebas.		

Prioridad y valor	Los ítems del backlog están priorizados por el Product Owner según el valor de negocio. Se confirma que los requisitos más importantes para el cliente se abordan primero, de acuerdo con la priorización establecida en Scrum.		
Claridad y entendimiento común	Las historias de usuario están redactadas de forma comprensible y han sido entendidas por el equipo. Cada historia sigue el formato establecido (por ejemplo, "Como rol quiero función para beneficio") y se discutió con el equipo en refinamiento o planificación para asegurar entendimiento común.		
Criterio INVEST	Las historias de usuario cumplen con los criterios INVEST (Independientes, Negociables, Valiosas, Estimables, Pequeñas, Testeables). Se evalúa que las historias estén bien formadas: no dependen entre sí estrechamente, aportan valor claro, pueden estimarse, tienen un tamaño manejable para un sprint y son verificables mediante criterios de aceptación.		
Consistencia y no duplicación	No existen historias de usuario duplicadas, solapadas o contradictorias entre sí. El backlog fue revisado para eliminar redundancias o conflictos: cada requerimiento aparece una sola vez y no entra en conflicto con otros.		
Trazabilidad y contexto	Cada historia de usuario está vinculada a un objetivo de producto o épica mayor que le da contexto. Se comprueba que las historias derivan de metas del proyecto (épicas o features) definidas por el Product Owner, garantizando alineación con la visión del producto.		
Revisión de stakeholders	El Product Owner y los stakeholders relevantes han revisado y aprobado la lista de requisitos. Se evidencian revisiones o demos iniciales donde el cliente/PO		

	confirma que el conjunto de historias refleja lo que necesita antes de avanzar al desarrollo.		
Requisitos no funcionales	Se han documentado requisitos de calidad (no funcionales) relevantes. Aspectos como rendimiento, seguridad, usabilidad, etc., están capturados ya sea en historias específicas, criterios de aceptación o como restricciones generales del backlog, asegurando su consideración en el diseño.		

Extreme Programming (XP)

Criterio de evaluación	Descripción	Cumple (Sí/No)	Observaciones
Historias de usuario simples	Los requerimientos están expresados como historias de usuario breves en tarjetas, con detalles aclarados mediante conversaciones directas con el cliente. Se verifica que las historias son comprensibles y deliberadamente ligeras en detalle escrito, ya que se complementan con la comunicación constante con el cliente (práctica XP de "las 3C": Card, Conversation, Confirmation).		
Cliente in situ y validación continua	El cliente (usuario experto) participa diariamente y ha confirmado que las historias de usuario reflejan sus necesidades reales. Se comprueba que el cliente ha revisado y ajustado los requerimientos directamente junto con el equipo, garantizando alineación constante con las expectativas del usuario final.		
Pruebas de aceptación definidas	Para cada historia de usuario el cliente ha definido o acordado pruebas de aceptación concretas. Existen criterios de prueba (ejemplos de uso o casos de prueba de alto nivel) escritos en lenguaje sencillo, que servirán para verificar que cada historia se implementó correctamente según los criterios del cliente.		

Priorización por valor	Las historias están priorizadas según el valor de negocio y el riesgo, tal como decidido en el “Planning Game”. Se corrobora que el orden en que se abordarán las historias fue establecido por el cliente y el equipo considerando qué entrega más valor primero y qué tan estimadas fueron (costo/esfuerzo).		
Alcance iterativo y adaptable	Solo se detallan las historias previstas para la iteración actual o próxima, permitiendo cambiar y agregar requisitos en iteraciones futuras sin impacto en documentación previa. El conjunto de requerimientos se mantiene flexible: no hay un documento rígido que necesite cambios mayores, sino que el plan se reevalúa en cada iteración, adaptándose a nuevas ideas o prioridades.		
Enfoque en lo necesario (YAGNI)	No se incluyen requisitos que el cliente no haya solicitado explícitamente para satisfacer una necesidad actual designcodetips.blogspot.com . Siguiendo el principio "No construir nada hasta necesitarlo", se revisa que no haya funcionalidades ‘por si acaso’: el alcance del SRS XP se limita a lo que aporta valor inmediato, evitando características superfluas.		
Requisitos de calidad implícitos manejados	Criterios como rendimiento, seguridad u otros atributos de calidad están siendo manejados ya sea mediante historias técnicas separadas o agregados a los criterios de aceptación. Aunque XP no produce un documento formal de requisitos no funcionales, se comprueba que el equipo y el cliente han discutido estas consideraciones y las han convertido en parte del “Definition of Done” o en historias dedicadas, para no pasarlas por alto.		
Coherencia con la	Los requerimientos encajan bajo una visión compartida o metáfora simple del sistema definida por el equipo. Se verifica que el conjunto de funcionalidades descritas tiene		

metáfora del sistema	un hilo conductor o tema común (metáfora) que facilita al equipo entender la arquitectura de alto nivel de forma unificada, en lugar de un documento extenso de diseño.		

Rational Unified Process (RUP)

Criterio de evaluación	Descripción	Cumple (Sí/No)	Observaciones
Formato y secciones completas	El documento SRS sigue el formato estándar (introducción, alcance, definiciones, requisitos funcionales, requisitos no funcionales, etc.). Se verifica que el SRS está organizado según una plantilla o estándar (p. ej. IEEE 830 o plantilla RUP), incluyendo todos los apartados requeridos para comprender el producto: propósito, alcance del sistema, referencias, descripción general, listado de requerimientos funcionales con sus descripciones detalladas, y sección de requerimientos no funcionales.		
Requisitos funcionales derivados de casos de uso	Cada requerimiento funcional está asociado a uno o más casos de uso identificados en el modelo de casos de uso del sistema. Se comprueba que el SRS lista todas las funcionalidades requeridas, a menudo estructuradas en torno a casos de uso. Los casos de uso relevantes del sistema están enumerados y enlazados con los requerimientos, y cada caso de uso tiene una descripción de sus flujos o escenarios en documentación complementaria.		
Requisitos no funcionales documentados	El SRS incluye todos los requisitos de calidad y restricciones técnicas del sistema en una sección dedicada. Se revisa que aspectos como rendimiento, seguridad, usabilidad, compatibilidad, etc., estén claramente especificados con criterios medibles donde sea posible. Ningún requisito de este tipo se deja		

	implícito; todos están escritos y categorizados apropiadamente.		
Identificación y trazabilidad	Cada requisito en el SRS está numerado o etiquetado de forma única, permitiendo rastrear su implementación en los diseños, código y pruebas. Se verifica que cada requerimiento tiene un identificador (por ejemplo, RF-1, RNF-3) y que existe una matriz de trazabilidad o referencia cruzada que vincula los requisitos con los casos de uso, módulos de diseño o casos de prueba, facilitando el seguimiento de cobertura.		
No ambigüedad y claridad	La redacción de los requisitos es clara, sin ambigüedades ni lenguaje interpretable. Se buscan términos vagos o subjetivos. Cada requisito está escrito en términos verificables (p.ej., usando "debe hacer X bajo condición Y" en lugar de expresiones generales). Si se encontraron siglas o términos técnicos, se definen en la sección de glosario del documento para evitar malentendidos.		
Consistencia interna	No existen conflictos entre diferentes requisitos dentro del SRS, ni duplicaciones. Se revisa que dos requisitos no se contradigan (p.ej., un requisito no anule a otro). También se comprueba que cada funcionalidad esté descrita una sola vez; si aparece en múltiples secciones, la información es consistente.		
Alineación con documentos de inicio	El contenido del SRS refleja fielmente la Visión del proyecto y los acuerdos con stakeholders obtenidos en las fases iniciales. Se comprueba que los objetivos y alcances definidos en el documento de visión o acta de requerimientos inicial se reflejan en los requisitos listados. No hay requisitos añadidos que no hayan sido		

	acordados con los interesados durante la fase de Incepción/Elaboración de RUP.		
Revisión y aprobación formal	El SRS ha sido revisado por las partes interesadas (clientes, analistas, equipo técnico) y cuenta con una aprobación formal o firma de conformidad. Se verifica la existencia de una sección de historial de revisiones o firmas, o al menos evidencia de que el documento fue sometido a una revisión técnica/formal (inspección de requisitos) antes de considerarse base para el diseño. Esto asegura que el SRS ha pasado por control de calidad de requerimientos.		
Control de cambios	Existe un procedimiento definido para gestionar cambios al SRS una vez aprobado (línea base establecida). Se indica que cualquier modificación futura de un requisito seguirá un proceso de gestión de cambios (por ejemplo, peticiones de cambio evaluadas por un comité o analista de requisitos), lo cual es típico en RUP para mantener la estabilidad del alcance una vez que se cierra la etapa de requisitos.		

Rapid Application Development (RAD)

Criterio de evaluación	**Descripción**	**Cumple (Sí/No)**	**Observaciones**
Captura rápida con usuarios	Los requisitos se recopilaron mediante talleres intensivos con usuarios y clientes (p. ej., sesiones JAD – Joint Application Design). Se verifica que el origen del SRS fue la colaboración directa con usuarios finales en sesiones enfocadas, donde aportaron sus necesidades y retroalimentación inmediata, característica clave de RAD para alinear rápidamente el producto con las expectativas del usuario.		

Enfoque en requisitos esenciales	El documento de requisitos se concentra solo en las funcionalidades clave acordadas, evitando incluir requerimientos no prioritarios en esta fase. Se revisa que el SRS no esté sobrecargado de detalles o características “deseables” de baja prioridad. RAD busca entregar rápidamente un núcleo funcional; por tanto, se documentan primero las características fundamentales que resolverán el problema del cliente.		
Validación con prototipos	Cada requisito del SRS ha sido validado a través de prototipos o modelos preliminares presentados a los usuarios. Se comprueba que para las funcionalidades listadas, el equipo creó maquetas o prototipos tempranos que los usuarios revisaron. Cualquier corrección o cambio sugerido a raíz de esas revisiones se reflejó en la versión actual de los requisitos, asegurando que el documento corresponde a lo que los usuarios realmente esperan.		
Flexibilidad y evolución	El SRS se considera un documento vivo que se actualiza fácilmente a medida que evolucionan las necesidades durante el desarrollo rápido. Se evidencia que el documento está preparado para cambios frecuentes: por ejemplo, los requisitos pueden reordenarse o ajustarse entre iteraciones RAD sin trámites burocráticos. La redacción permite añadir o modificar requerimientos de forma ágil conforme surjan nuevas ideas o necesidades en el proceso de desarrollo iterativo.		
Prioridad y versionado iterativo	Los requisitos están etiquetados o agrupados según versiones o entregas incrementales del producto. Se verifica que el SRS indica qué requisitos serán abordados en cada ciclo o iteración de		

	desarrollo rápido. Esto puede verse en notas que marcan ciertos requisitos como parte del “Entregable 1”, “Prototipo 2”, etc., lo cual guía el desarrollo por fases y mantiene al cliente informado sobre qué esperar en cada entrega.		
Integración de requisitos no funcionales mínimos	Se han identificado las restricciones críticas (p.ej., performance básico, entornos tecnológicos) necesarias para guiar el desarrollo, aunque el enfoque esté en funcionalidad. Si bien RAD privilegia la funcionalidad tangible y la interfaz, se comprueba que no se pasaron por alto requerimientos técnicos indispensables. Por ejemplo, si el sistema debe operar en web y móvil, o manejar cierto volumen de datos, tales requerimientos se mencionan para evitar re-trabajos más adelante, manteniendo el producto alineado con su contexto de uso.		
Colaboración equipo-usuario continua	El SRS refleja decisiones tomadas conjuntamente por desarrolladores y usuarios durante iteraciones de diseño y construcción. Más que un contrato fijo, el documento de requisitos sirve como memoria de las decisiones de alcance tomadas en colaboración. Se revisa que el lenguaje y nivel de detalle sean entendibles para todos (no excesivamente técnico), facilitando que tanto el equipo como los usuarios puedan revisarlo y modificarlo juntos en siguientes ciclos.		
Documentación complementaria ligera	Además del SRS textual, se usan apoyos visuales (diagramas de flujo simples, pantallazos de prototipos) para clarificar requisitos cuando aplica. Se verifica que, de ser necesario, el equipo adjuntó o referenció elementos visuales (por ejemplo, pantallas diseñadas en la fase de “User Design”) que ayudan a		

	entender mejor ciertos requisitos, reduciendo la necesidad de largas descripciones textuales y aprovechando la naturaleza visual e iterativa de RAD.		

Validación de Modelos UML (Casos de uso, Clases, Actividades)

Scrum

Criterio de evaluación	**Descripción**	**Cumple (Sí/No)**	**Observaciones**
Casos de uso alineados con historias	Los diagramas de casos de uso representan las funcionalidades del Product Backlog y actores identificados en el proyecto. Se comprueba que cada historia de usuario importante del backlog corresponde a un caso de uso en el diagrama, asegurando que el modelado refleja lo que realmente se está construyendo. Los actores del diagrama coinciden con los roles de usuarios definidos en los requisitos.		
Simplicidad en el modelado	Solo se han modelado los procesos y entidades necesarios para comprender y planificar el trabajo del sprint actual. Se revisa que los diagramas UML no contengan detalles innecesarios: por ejemplo, el diagrama de clases incluye únicamente las clases relevantes para las historias de usuario en curso, evitando sobre-diseñar partes del sistema que aún no se van a desarrollar (principio YAGNI aplicado al diseño).		
Colaboración en la creación de modelos	El equipo de desarrollo participó conjuntamente en la elaboración o actualización de los diagramas UML durante la planificación o sesiones de diseño. Se verifica que los modelos no fueron creados en aislamiento por una sola persona, sino que se discutieron en equipo (por ejemplo, en una reunión de planificación de sprint o diseño técnico		

	breve), promoviendo entendimiento compartido y detectando errores de diseño tempranamente.		
Actualización iterativa	Los modelos UML se mantienen actualizados al finalizar cada sprint o cuando hay cambios significativos en el diseño. Se comprueba que la documentación de los diagramas es viva: si una funcionalidad fue agregada o cambiada en un sprint, los diagramas de casos de uso, clases o actividades reflejan ese cambio. No se encuentran diagramas obsoletos que no correspondan al estado actual del sistema.		
Corrección y estandarización de notación	Los elementos UML usados siguen la notación correcta y estándar (actores, casos de uso, clases, relaciones, nodos de actividad, etc.). Aunque Scrum no enfatiza la documentación formal, se verifica que los diagramas realizados sean técnicamente correctos en su sintaxis UML para evitar malentendidos. Por ejemplo, en el diagrama de clases se usan correctamente las asociaciones con multiplicidades, herencias si las hay, y en diagramas de actividad se representan bien las decisiones y flujos.		
Consistencia entre modelos	Si se utilizan múltiples tipos de diagramas (casos de uso, clases, actividades), la información es consistente entre ellos. Se revisa que un proceso descrito en un diagrama de actividad corresponda a un caso de uso del diagrama de casos de uso, y que las entidades involucradas aparezcan en el diagrama de clases. No debe haber contradicciones; por ejemplo, un caso de uso en el que interviene cierta clase debe reflejarse adecuadamente en el modelo de clases.		

Enfoque en valor de negocio	Los diagramas de actividad se enfocan en ilustrar los flujos de trabajo de valor para el usuario, evitando diagramar procesos internos irrelevantes para la funcionalidad visible. Se verifica que los modelos reflejen casos de uso orientados al valor (lo que el usuario hace y obtiene). Cualquier diagrama de actividad elaborado corresponde a un escenario de caso de uso significativo (flujo principal o alternativo importante) y no a detalles de implementación técnica de bajo nivel que no aportan al entendimiento funcional en esta etapa.		
Verificación con Product Owner	Los modelos (especialmente casos de uso) han sido presentados al Product Owner o stakeholders para validar que representan correctamente los requisitos. Se comprueba que el equipo ha utilizado los diagramas como una herramienta de comunicación: por ejemplo, mostrando el diagrama de casos de uso al Product Owner para asegurar que no falta ningún caso importante. Esto garantiza que el modelado está alineado con la visión del cliente y que sirve como confirmación de entendimiento antes de desarrollar.		

Extreme Programming (XP)

Criterio de evaluación	**Descripción**	**Cumple (Sí/No)**	**Observaciones**
Diseño simple y enfocado	Los diagramas (p.ej., un diagrama de clases simple) representan únicamente la estructura necesaria para implementar las historias de usuario actuales, sin añadir complejidad adicional. Se verifica que el diseño refleja la solución más simple posible para cumplir los requisitos en curso (principio de diseño simple en XP). No		

	hay clases, métodos o pasos en diagramas de actividad que no correspondan a una necesidad inmediata del usuario		
Uso de técnicas ligeras (CRC)	En lugar de modelos UML detallados, se han utilizado técnicas ágiles como tarjetas CRC para identificar responsabilidades de clases y colaboraciones de manera rápida. Se comprueba que el equipo pudo discutir y asignar responsabilidades clave a las clases usando tarjetas CRC u otro método ligero, asegurando que entienden el diseño sin necesidad de documentación UML extensa. Si se plasmaron en un diagrama de clases, este es esbozado con la misma simplicidad y enfoque en responsabilidades claras.		
Correspondencia con el código y pruebas	Cualquier diagrama realizado corresponde al estado actual del código y ha sido validado mediante pruebas unitarias/de aceptación. En XP el código es la principal fuente de verdad. Se revisa que si se dibujó un diagrama de clases o de secuencia para planificar una funcionalidad, este coincide con lo que luego se programó. Además, las pruebas automatizadas pasan exitosamente, indicando que el diseño (implícito en el código) soporta correctamente los casos de uso previstos.		
Evolución mediante refactorización	El diseño (y sus diagramas si existen) se han ajustado después de implementar historias, eliminando duplicaciones o mejorando la estructura. Se verifica que el equipo aplicó refactorización una vez que las nuevas funcionalidades estuvieron en marcha, simplificando los modelos. Por ejemplo, si se descubrió que dos clases podían unificarse o que una función era redundante, los		

	diagramas se actualizaron o las tarjetas CRC se ajustaron para reflejar la nueva y mejorada composición del sistema.		
Metáfora del sistema mantenida	El desarrollo sigue una metáfora común del sistema y los nombres de casos de uso, clases y actividades en los modelos reflejan esa metáfora de forma coherente. Se comprueba que los elementos del diseño comparten una visión conceptual simple acordada. Por ejemplo, si la metáfora elegida fue "Librería", las clases y casos de uso usan terminología de ese dominio (Libro, Estante, Préstamo) consistentemente. Esto ayuda a guiar el diseño sin necesidad de extensos documentos arquitectónicos.		
Comunicación continua	Los modelos (o bocetos) se discutieron en tiempo real durante la programación en pareja o las sesiones de diseño colectivo. Se evidencia que las decisiones de diseño fueron tomadas en grupo y en el momento, en lugar de preparar diagramas formales por adelantado. Por ejemplo, ante un problema, los desarrolladores dibujaron rápidamente un diagrama de actividad o de secuencia en una pizarra para aclarar el flujo, resolvieron el diseño y continuaron codificando. Dichos bocetos se entienden como guías temporales y no necesariamente quedan como documentación permanente.		
Ausencia de burocracia en modelos	No se exigió aprobación formal ni documentación exhaustiva de los diagramas; estos se crean y descartan conforme se necesiten. Se comprueba que el proceso de modelado fue ágil: si un diagrama dejó de ser útil tras implementarse el código, no se dedicó tiempo a mantenerlo. La validación del diseño proviene más de las pruebas y la		

	funcionalidad entregada que de revisiones documentales. Esto se alinea con la filosofía XP de valorar el software funcionando por encima de la documentación.		
Correctitud básica de la notación utilizada	Si se emplearon diagramas UML, se usaron correctamente los elementos esenciales para evitar confusiones. Aunque los modelos en XP suelen ser informales, se revisa que cualquier diagrama de clases o actividad dibujado use símbolos comprensibles (p.ej., clases con nombres significativos, flechas de asociación o dependencias claras). Esto garantiza que, incluso con su sencillez, los diagramas comunicaron correctamente las ideas de diseño en el momento.		

Rational Unified Process (RUP)

Criterio de evaluación	**Descripción**	**Cumple (Sí/No)**	**Observaciones**
Modelo de Casos de Uso completo	Se han identificado y modelado todos los casos de uso del sistema, mostrando en el diagrama todos los actores y sus interacciones con los casos de uso. Se verifica que el diagrama de casos de uso cubre todas las funcionalidades descritas en el SRS. Cada funcionalidad o requisito funcional del sistema corresponde a al menos un caso de uso en el modelo. No faltan casos de uso importantes y cada actor (humano o sistema externo) que interactúa con el sistema está representado, garantizando cobertura completa de requisitos.		
Detalle de casos de uso	Cada caso de uso del diagrama tiene un documento o descripción de caso de uso asociado con sus flujos de eventos (principal y alternos) y condiciones. Aunque esto va más allá del diagrama		

	UML en sí, en RUP se comprueba que por cada caso de uso modelado existe una especificación escrita que detalla cómo se desarrolla la interacción. Esta verificación asegura que el diagrama no es superficial: hay sustancia detrás de cada óvalo, describiendo escenarios, excepciones, pre y post condiciones, lo que facilita la derivación de clases y actividades correspondientes.		
Diagrama de clases de análisis/diseño	El modelo de clases incluye las clases de análisis clave (entidades, control, interfaz) identificadas a partir de los casos de uso. Se revisa que el diagrama de clases represente la estructura estática necesaria para cumplir los casos de uso. Las clases de entidad reflejan los conceptos del dominio (objetos principales del sistema), las clases de control gestionan la lógica de los casos de uso, y las clases de interfaz (boundary) manejan la interacción con los actores externos, siguiendo las recomendaciones de RUP para diseño guiado por casos de uso.		
Relaciones y cardinalidades correctas	En el diagrama de clases, las asociaciones, agregaciones o composiciones entre clases están correctamente establecidas, incluyendo multiplicidades y posible uso de generalizaciones o realizaciones de interfaces cuando corresponde. Se comprueba la exactitud técnica del modelo de clases: por ejemplo, si un caso de uso implica que un Usuario puede realizar muchas Órdenes, se refleja con una asociación uno-a-muchos entre las clases Usuario y Orden con multiplicidades 1..∗1..*1..∗. Cualquier herencia necesaria se modela para evitar duplicación de atributos. Las relaciones reflejan fielmente las interacciones		

	requeridas por los casos de uso sin inconsistencias.		
Diagramas dinámicos por flujo	Se han elaborado diagramas de actividad o secuencia para representar los flujos principales de los casos de uso más críticos. Se verifica que para los casos de uso de mayor relevancia o complejidad, existen diagramas que modelan su comportamiento dinámico (p.ej., un diagrama de actividad mostrando el flujo de trabajo del caso de uso completo, o diagramas de secuencia mostrando la interacción temporal entre objetos en un escenario específico). Esto asegura que el equipo analizó la secuencia de pasos necesarios y la distribución de responsabilidades entre objetos para lograr cada funcionalidad.		
Consistencia entre modelos	Hay coherencia entre el diagrama de casos de uso, los diagramas de actividad y el diagrama de clases. Se comprueba que cada paso importante de un flujo de caso de uso pueda rastrearse a operaciones o interacciones en el diagrama de clases o secuencia. Por ejemplo, una acción en un diagrama de actividad debería corresponder a un método en alguna clase del diagrama de clases. No deben existir pasos en los flujos que no tengan respaldo en la estructura de clases, ni clases con funcionalidades que no estén motivadas por algún escenario de caso de uso.		
Uso correcto de la notación UML	Todos los diagramas siguen la notación UML estándar sin errores (actores bien ubicados fuera de los límites del sistema, relaciones de inclusión/extensión en casos de uso si aplican, símbolos de inicio/fin en actividades, etc.). Se revisa detalladamente la sintaxis UML: por ejemplo, que las flechas de comunicación en el diagrama de casos de uso estén bien orientadas, que en el diagrama de		

	actividad las decisiones tengan ramas rotuladas, y que cada clase en el diagrama de clases tenga nombre sustantivo y, de ser posible, lista de atributos y operaciones relevantes. Un modelo bien formado es crucial en RUP para evitar malinterpretaciones durante la implementación.		
Validación por parte del equipo	Los modelos UML han sido revisados en una inspección técnica o walkthrough con analistas, diseñadores y posiblemente usuarios. Se evidencia que el equipo realizó al menos una revisión formal de los diagramas: por ejemplo, reuniones de revisión de diseño en las que se recorrieron los casos de uso y sus realizaciones en clases/actividades, asegurando que todos están de acuerdo con la solución propuesta. En RUP es común hacer revisiones en la fase de Elaboración para aprobar la arquitectura; esta lista verifica que dichas revisiones de calidad de modelos se hayan llevado a cabo satisfactoriamente.		
Trazabilidad con requisitos	Cada elemento del modelo (caso de uso o clase clave) puede trazarse a uno o más requisitos del SRS, asegurando que el diseño cubre todo el alcance definido. Se comprueba la existencia de referencias cruzadas: por ejemplo, cada caso de uso indica qué requerimientos funcionales cubre, y cada requisito no funcional crítico (p.ej., seguridad) está reflejado en decisiones de diseño (como la inclusión de clases de seguridad o restricciones en los diagramas de actividad). Esto garantiza que no se diseñó nada fuera de alcance ni se olvidó implementar algo requerido.		

Rapid Application Development (RAD)P

Criterio de evaluación	Descripción	Cumple (Sí/No)	Observaciones
Modelado inicial de casos de uso	Se elaboró un diagrama de casos de uso temprano que identifica las funcionalidades fundamentales a desarrollar rápidamente. Se verifica que el equipo, junto con los usuarios, diagramó de manera ágil los casos de uso principales durante la fase inicial (Requirements Planning) para establecer el alcance. Este diagrama sirve como mapa general de qué funciones el sistema deberá cubrir, proporcionando un entendimiento común antes de construir prototipos.		
Diagramas de actividad centrados en el usuario	Para los procesos clave identificados, se crearon diagramas de actividad que muestran el flujo de trabajo del usuario a través del sistema. Se comprueba que el equipo representó visualmente cómo el usuario realizará tareas en el sistema (p. ej., flujo de un pedido, proceso de registro, etc.), a fin de validar con los usuarios que se entendieron bien sus procedimientos. Estos diagramas, creados en talleres, ayudan a descubrir requisitos mientras se dibujan, sirviendo como base para diseñar pantallas en el prototipo.		
Diseño de clases incremental	El diagrama de clases fue construyéndose a medida que se definían los componentes en cada iteración, empezando por las clases principales necesarias para el primer prototipo. Se revisa que el modelo de clases no pretende cubrir todo el sistema desde el inicio, sino que va creciendo con cada ciclo. Por ejemplo, tras la primera iteración, puede haber clases representando entidades centrales del dominio que se trabajaron (con atributos básicos); en iteraciones posteriores se agregan nuevas clases o se detallan más		

	atributos a medida que surgen nuevos requisitos en los siguientes prototipos.		
Actualización tras feedback	Los modelos UML se han ajustado después de cada demostración de prototipo según la retroalimentación del usuario. Se verifica que los cambios solicitados por los usuarios durante las revisiones de prototipo se reflejaron en los diagramas. Si un caso de uso sufrió modificaciones o un nuevo caso de uso apareció, el diagrama de casos de uso fue actualizado. Si se identificaron nuevas clases o cambios en relaciones a partir del feedback, el modelo de clases se revisó consecuentemente. Esto mantiene la documentación de diseño alineada con el estado actual del desarrollo, iteración tras iteración.		
Compleción progresiva de modelos	Los diagramas no están completos desde el inicio, pero al final del proceso RAD los modelos UML abarcan la mayor parte del sistema desarrollado. Se comprueba que, aunque los primeros diagramas pudieran ser parciales, el equipo fue completando los modelos a medida que más partes del sistema se implementaban. Al concluir las iteraciones principales, el diagrama de casos de uso debería reflejar todas las funciones construidas, el diagrama de clases todas las entidades realmente usadas, etc., funcionando como documentación final del producto resultante.		
Simplicidad y claridad visual	Los diagramas creados son sencillos, limpios y entendibles rápidamente por todos los participantes del proyecto. Se revisa que, en favor de la rapidez, los modelos no se hayan sobrecargado con detalles técnicos innecesarios. Por ejemplo, los diagramas de clases quizás no listan todos los métodos, solo los atributos y relaciones principales para comprender la estructura. Lo importante		

	es que cualquier miembro del equipo o usuario pueda leer el diagrama y captar la idea del sistema sin dificultad.		
Participación conjunta en modelado	Usuarios y desarrolladores colaboraron en la elaboración de los modelos durante las sesiones de diseño conjunto (JAD). Se evidencia que los diagramas fueron fruto de un esfuerzo colaborativo: durante los talleres JAD, los participantes discutían y acordaban los elementos que aparecían en los diagramas. Esta co-creación asegura que los modelos reflejen correctamente los requerimientos del usuario, ya que se construyeron con su propia contribución en tiempo real.		
Herramientas y rapidez	Se emplearon herramientas de modelado que permitieron generar y modificar diagramas rápidamente para seguir el ritmo de cambios. Se verifica que el equipo utilizó medios eficientes (software CASE sencillo o incluso pizarras y luego digitalización) para no retrasar el proceso. Cualquier diagrama necesario se pudo crear "sobre la marcha" durante una reunión. Si se usó una herramienta, los modelos podían exportarse hacia el prototipo o documentación inmediatamente, manteniendo la velocidad característica de RAD.		
Verificación básica de consistencia	A pesar de la velocidad, los diferentes diagramas (casos de uso, clases, actividades) siguen siendo coherentes entre sí en lo esencial. Se comprueba que los nombres de actores y casos de uso usados en los diagramas de actividad coinciden con los del diagrama de casos de uso, y que las entidades mencionadas en actividades aparecen en el diagrama de clases. No se permitieron incoherencias mayores, ya que incluso en RAD la alineación entre modelos evita		

	confusiones durante la construcción rápida del software.		