Dinámica del análisis

A diferencia de la educción de requisitos, la actividad de análisis posee una cierta estructura interna mostrada en la Figura 1. Nótese que, en la Figura 1, la *Lista preliminar de requisitos* es el producto resultante de la educción.

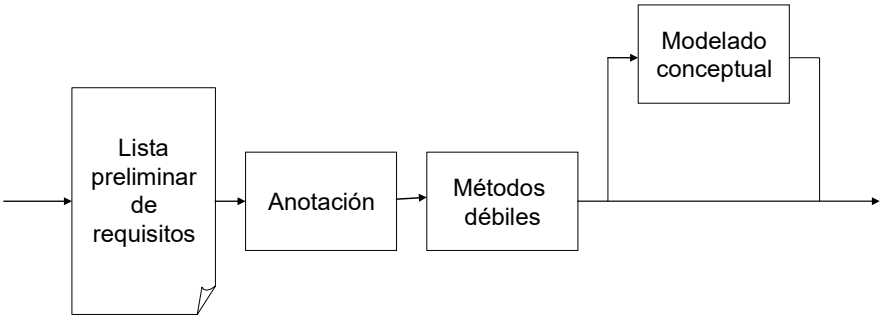

Figura 1. Proceso de análisis

Las tareas de las que se compone el análisis son las siguientes:

- **Anotación de los requisitos**.
- **Aplicación de los métodos débiles de análisis**. Los métodos débiles de análisis son un conjunto de técnicas simples que permiten comprobar rápidamente la calidad de los requisitos obtenidos durante la educción. Las técnicas más utilizadas son el checklist de análisis y la matriz de interacción.
- Finalmente, y en tercer lugar, si la complejidad del sistema software lo aconseja, es posible utilizar el **modelado conceptual** para comprobar la calidad general de los requisitos del software.

[2] Lo mismo sucede con el resto de las anotaciones.

Clasificación, o anotación, de requisitos

Anotar los requisitos significa asignar a éstos ciertos valores de importancia, prioridad, etc., de tal modo que se facilite la gestión de requisitos posterior. A efectos prácticos, la anotación se realiza en tres fases:

- **Anotar los requisitos por identificación y versión**. El analista debe realizar esta anotación en todos los proyectos de desarrollo, debido a su importancia para la trazabilidad y el control de versiones.
- **Decidir qué tipos de anotación suplementaria son necesarios**. En muchos casos, las características del proyecto no exigen anotaciones suplementarias. Por ejemplo, si todos los requisitos están bien determinados y todos ellos van a implementarse, es ocioso anotar por importancia o estabilidad. No obstante, en otros casos, realizar anotaciones suplementarias puede ser vital para el éxito del proyecto.
- **Realizar las anotaciones suplementarias**. Estas anotaciones deben ser realizadas por el analista con el concierto de los clientes y usuarios. De hecho, en la mayoría de las ocasiones, los valores de las anotaciones suplementarias se obtienen durante la misma actividad de educción.

Métodos débiles de análisis

Existen, principalmente, dos métodos débiles de análisis: el **checklist de análisis y la matriz de interacción**. La denominación de métodos débiles surge del hecho de que, al contrario que los modelos conceptuales, aquellos son válidos en todo tipo de sistema software y configuración de proyecto. Esto es, los métodos débiles pueden usarse siempre, mientras que los modelos conceptuales, dependiendo del tipo de sistema, pueden ser más o menos útiles.

A. Checklist de análisis

Un checklist de análisis es, simplemente, *un conjunto de preguntas* que el analista debe considerar *para cada requisito individual*. Estas preguntas están relacionadas con alguno de los siguientes atributos de calidad:

- Independencia del diseño
- Concisión
- Realizabilidad
- Externamente consistente
- Ambigüedad
- Verificabilidad

El propósito del checklist es asegurar que el analista ha descrito adecuadamente cada requisito de la lista preliminar de requisitos. En caso de que el analista detectase alguna carencia, éste debería corregir el requisito defectuoso de forma inmediata.

Para la aplicación de este método débil, existen algunos (pocos) checklists predefinidos. La Tabla 1 muestra un checklist propuesto en [Kotonya and Sommerville, 1998] y que puede ser útil en una gran diversidad de proyectos.

Finalmente, es necesario indicar que, a medida que el analista gana en experiencia, éste acostumbra a redactar de una forma bastante adecuada la lista preliminar de requisitos, por lo que no es posible detectar carencias en los requisitos con checklists como el mostrado en la Tabla 1. En este caso, es necesario utilizar modelado conceptual.

Atributo de calidad a considerar	**Pregunta**
Independencia del diseño	• ¿La lista de requisitos anticipa el diseño o incluye información de implementación?
Concisión	• ¿Cada requisito es simple o, por el contrario, podría dividirse en varios requisitos? • ¿Existe algún requisito que no parezca añadir ninguna información útil acerca del sistema a desarrollar?
Realizabilidad	• ¿Es realizable el requisito en la plataforma de implementación? • ¿Existe algún requisito irrealizable con la tecnología actual? **NOTA**: Para responder a esta pregunta, es necesario conocer los aspectos técnicos de la plataforma donde se implementará el sistema.
Consistencia externa	• ¿Existe algún requisito que contradiga requisitos organizativos explícitamente formulados?
Ambigüedad	• ¿Es posible interpretar de varias formas un requisito?
Verificabilidad	• ¿Es posible idear algún caso de prueba que permita establecer que el requisito NO SE CUMPLE?

Tabla 1. Checklist de análisis

Problemas en la aplicación del checklist de análisis

El problema más común que surge durante la utilización de los checklist de análisis es la imposibilidad de localizar defectos en los requisitos. Esto sucede cuando el analista aplica el checklist, pero no detecta ningún requisito defectuoso.

Lo anterior puede estar motivado por dos factores: (1) el analista posee mucha experiencia y ha confeccionado adecuadamente la lista preliminar de requisitos y, lo que es más probable, (2) el analista no es capaz de descubrir defectos en la lista preliminar de requisitos por que ha sido él mismo el que la ha confeccionado.

Existe una máxima bastante antigua de la validación del software, y dice más o menos que una persona no puede probar un programa que él mismo ha construido. Con los requisitos sucede

exactamente lo mismo: no es eficaz, en muchas ocasiones, intentar localizar defectos en los requisitos escritos por uno mismo.

En estas circunstancias, lo mejor es que el checklist de análisis sea aplicado por una persona distinta que el analista que confeccionó la lista preliminar de requisitos (un compañero de trabajo, por ejemplo). Ésta otra persona tendría dos responsabilidades:

- Localizar defectos en los requisitos
- Comunicar estos defectos al analista, junto con sugerencias acerca de cómo solucionar dichos defectos

Esto es, la persona que aplica el cheklist tiene la responsabilidad de identificar errores y comunicárselos al analista, pero no la de corregirlos por sí mismo. Muy al contrario, esta es responsabilidad del analista, dado que es el que mejor conoce el sistema software a desarrollar. Los errores y las sugerencias de solución acostumbran a plasmarse en un documento como el mostrado en la Tabla 2, denominado ***Lista de errores y acciones recomendadas*.**

Nº de requisito	**Defectos detectados**	**Acciones recomendadas**
1	Error de estilo, que lleva a ambigüedad	Modificar el texto del requisito de tal forma que diga algo como “El sistema deberá permitir el registro de los fondos bibliográficos”
2	Idem	Idem
11	Ambigüedad	Precisar la duración de las reservas
12	Concisión	No se han identificado diferencias entre profesores y alumnos a todo lo largo de la lista de requisitos. Este requisito se debería eliminar, ya que proporciona ningún tipo de información relevante.
15	Realizabilidad	El sistema no puede realizar automáticamente los préstamos. Quizás se refiere el requisito a que debe proporcionar el máximo de automatización? Precisar, en este caso, o eliminar por irreal.
17	Concisión	Separar lo referido a libros prestados de lo referido a libros reservados.

Tabla 2. Lista de errores y acciones recomendadas
(a modo de ejemplo)

B. Matriz de interacción

Una matriz de interacción es, simplemente, una matriz de doble entrada donde se *cruzan todos los requisitos entre sí*, tal y como muestra la Tabla 3. Por cruzar, debe entenderse que para cualesquiera dos requisitos r_1 y r_2, se debe comprobar si:

- r_1 se solapa con r_2, esto es, r_1 trata aspectos del sistema también tratados en r_2. Ello, de ser cierto, daría lugar a problemas de redundancia. Esto es lo que se ha indicado con una S en la Tabla 3.
- r_1 está en conflicto con r_2, esto es, r_1 y r_2 son contradictorios. Ello, lógicamente, da lugar a problemas de consistencia interna. Esto es lo que se ha indicado con una C en la Tabla 3.

	r_1	r_2	...	r_n
r_1		C		
r_2				S
...				
r_n				

Tabla 3. Matriz de interacción

En el caso de que dos requisitos r_1 y r_2 estén solapados, la solución es simple: Dado que se puede producir un problema de redundancia, el analista debe condensar los requisitos r_1 y r_2 en un nuevo requisito r_{1+2}, que refleje los contenidos de los requisitos r_1 y r_2. Condensar los requisitos de este modo es interesante, ya que produce un efecto positivo aparte de la disminución de redundancia: los requisitos tienden a dejar de expresarse como requisitos de usuario y se describen como requisitos del sistema.

Reutilizando el ejemplo usado en la sección 6.1, los requisitos 'El sistema deberá generar facturas' y 'Las facturas deberán generarse mensualmente' están solapados (tratan el mismo aspecto del sistema) y, por lo tanto, producen redundancia. Al mismo tiempo, ambos están expresados claramente como requisitos de usuario.

Para eliminar la redundancia producida por los requisitos anteriores, es necesario condensar éstos. El resultado podría ser el siguiente:

- El sistema deberá generar facturas.
- Las facturas se generarán mensualmente.

Obviamente, lo anterior está expresado claramente como un requisito del software.

En el caso de que dos requisitos r_1 y r_2 estén en conflicto, la solución es más compleja. Salvo que el conflicto resida en algún problema de expresión, o algún defecto de orden menor en la lista preliminar de requisitos, lo más probable es que su origen esté en conflictos en la concepción del sistema software por parte de los usuarios. Este tipo de defecto no puede ser resuelto por el analista en solitario: éste deberá acudir a los usuarios que han generado el conflicto y negociar con ellos hasta que el conflicto se resuelta, tal y como se indica en la sección 6.7.

Problemas en la aplicación de la matriz de interacción

Existen tres problemas principales en la aplicación del método de la matriz de interacción.

- **El ya mencionado**, acerca de la imposibilidad del analista de verificar sus propias creaciones. La solución, en este caso, sería la misma que la indicada anteriormente (permitir que la matriz de interacción fuese aplicada por otra persona y que ésta detectase los defectos en los requisitos y los plasmase en un alista de errores y acciones recomendadas).
- **Cómo detectar conflictos y solapamientos**. Esto es, dados dos requisitos r_1 y r_2, ¿cómo puede saber el analista, o la persona que esté aplicando la matriz de interacción, si ambos requisitos son conflictivos o están solapados?
  Esta pregunta es, no obstante, de fácil respuesta. Para poder aplicar la matriz de interacción, el analista, o la persona que lo substituye, debería poseer un buen conocimiento de la aplicación a construir. Sólo de este modo es posible no cometer errores, por omisión o comisión, a este respecto.
  Finalmente, si poseyendo un buen conocimiento de la aplicación, el analista, o la persona que lo substituye, no puede juzgar si dos requisitos r_1 y r_2 son conflictivos o están solapados[3], lo preferible es considerar que existe un problema potencial entre los requisitos r_1 y r_2 y estudiar éstos más detenidamente.
- **Cómo manejar grandes conjuntos de requisitos**. La única solución a este problema es descomponer el sistema software en varios subsistemas, con la finalidad de reducir la complejidad del análisis de cada uno de ellos.