

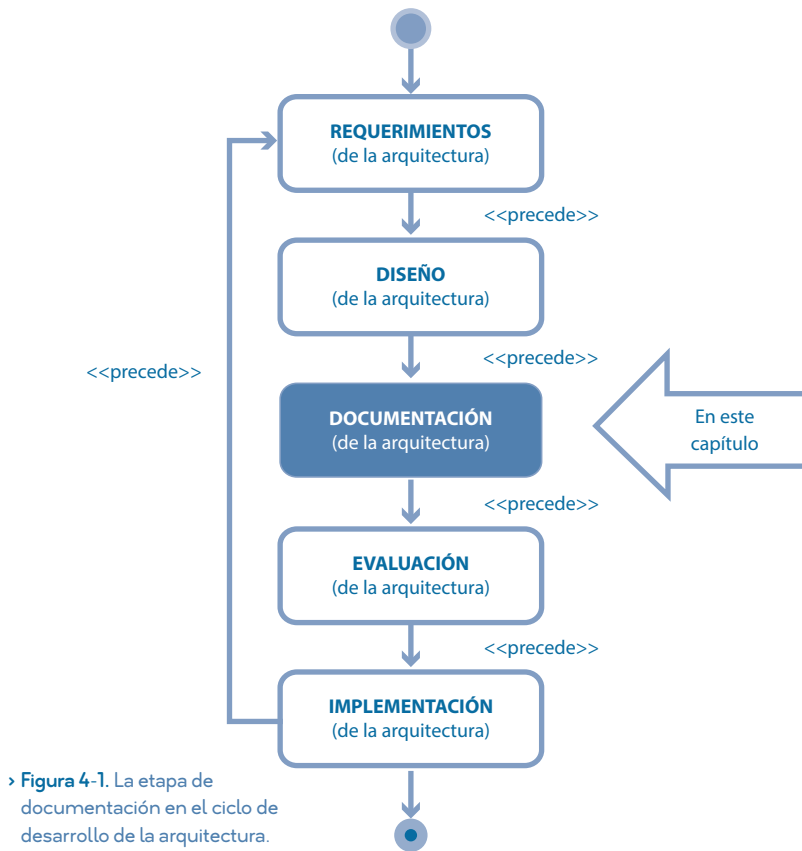
DOCUMENTACIÓN: COMUNICAR LA ARQUITECTURA

En el capítulo anterior explicamos que, durante la etapa de diseño, el arquitecto traduce los *drivers* en las estructuras arquitectónicas que conforman el sistema. Sin embargo, todo este esfuerzo sería inútil si, por ejemplo, durante la implementación de estas los desarrolladores tuvieran retrasos frecuentes porque no entienden completamente dichas estructuras.

En este capítulo describiremos la etapa de documentación de la arquitectura, resaltada en la figura 4-1. La etapa se enfoca en la elaboración de documentación con la que se comuniquen de manera eficiente las estructuras de la arquitectura a los diversos interesados en el sistema.

Describiremos de manera inicial qué se entiende por documentar en un contexto general. Después trasladaremos esta noción al ámbito de la *arquitectura de software* y discutiremos varias razones importantes para documentar una arquitectura. Presentaremos posteriormente los conceptos relevantes en esta etapa, las notaciones, así como algunos de los enfoques más conocidos que pueden utilizarse para soportar las actividades en esta etapa.

Concluiremos el capítulo enumerando una serie de recomendaciones útiles para generar documentación arquitectónica de buena calidad.



4.1 ¿QUÉ SIGNIFICA DOCUMENTAR?

En prácticamente cualquier entorno, conocer acerca de las políticas, procesos y productos que empleamos en nuestras actividades diarias es fundamental para la realización correcta de estas. En consecuencia, tener acceso a fuentes documentales adecuadas que describan la información necesaria resulta muy útil comparado con tener que extraerla directamente de las personas que la conocen o, peor aún, no tener esta información.

En términos generales, la palabra documentar se asocia al proceso de elaborar un documento, o conjunto de documentos, que tienen como propósito comunicar información relevante de un proceso, producto o entidad. Además de comunicarla, documentar permite que la información persista. Esto es muy importante si pensamos que en muchas ocasiones información relevante para soportar nuestras actividades laborales diarias no se documenta y es conocida solo por algunas personas dentro de la organización. Cuando ellas dejan de laborar en la organización, recuperar la información puede ser complicado o quizás imposible.

Ejemplos comunes de documentación incluyen las guías de usuario, manuales de políticas y procedimientos, así como tutoriales, documentos técnicos o guías de referencia rápida. Con los avances actuales en materia de tecnologías de la información, la documentación no solo puede estar plasmada en un conjunto de papeles. Medios como el video, los documentos electrónicos o los *wikis* se utilizan en la actualidad para plasmarla. Dependiendo del dominio o contexto de la organización, también se pueden variar las notaciones utilizadas para especificar la información en todos estos medios.

4.2 DOCUMENTACIÓN EN EL CONTEXTO DE ARQUITECTURA DE SOFTWARE

En el capítulo 1 mencionamos que la *arquitectura de software* es el término utilizado para referirse a las estructuras que conforman un sistema. En el capítulo 3 explicamos que durante la etapa de diseño, y con base en la información obtenida durante la fase de requerimientos, el arquitecto define esas estructuras mediante la toma de decisiones de diseño. De esta forma, en el contexto del ciclo de desarrollo de la *arquitectura de software*,

la etapa de documentación se centra en la generación de documentos que describen las estructuras de la arquitectura con el propósito que esta información pueda ser comunicada de manera eficiente a los diversos interesados en el sistema.

Aunque en la figura 4-1 la etapa de documentación de la arquitectura aparece después de la de diseño, en la práctica es difícil imaginar que se lleva a cabo estrictamente después de que todo el diseño arquitectónico está terminado. Muchas de las actividades en la etapa de diseño se realizan de manera concurrente con actividades de esta. Un ejemplo es que durante el diseño se hace uso de herramientas de modelado al crear las estructuras; estos modelos son una parte importante de la documentación.

4.3 RAZONES PARA DOCUMENTAR LA ARQUITECTURA

Al inicio del capítulo mencionamos que, pese a haber diseñado la mejor arquitectura, todo este trabajo puede ser poco productivo si personas como los diseñadores y desarrolladores del sistema tienen problemas al realizar sus tareas porque no existe documentación que la describa y, en cambio, hay información al respecto que no les ha sido comunicada. En la presente sección abundamos sobre estas y algunas otras razones por las cuales es importante documentar la arquitectura:

1. Mejorar la comunicación de información sobre la arquitectura.
2. Preservar la información sobre la arquitectura.
3. Guiar la generación de artefactos en otras fases del ciclo de vida.
4. Proveer un lenguaje común entre diversos interesados en el sistema.

4.3.1 Mejorar la comunicación de información sobre la arquitectura

Es claro que una vez diseñada la arquitectura deberá presentarse en algún momento a los diversos interesados en el sistema. Si bien es cierto que durante su diseño algunos modelos preliminares o bocetos pudieron haber sido generados, y el arquitecto podría emplearlos como apoyo para clarificar información durante la presentación de la arquitectura, es posible que su utilidad sea limitada pues podrían estar especificados en una notación no estándar o podrían contener información muy general. En este caso, el arquitecto tendría que comunicar de forma verbal la información faltante.

Una de las ventajas de la comunicación verbal es que es inmediata; sin embargo, se sabe ampliamente que esta dista de ser efectiva siempre. La efectividad de la comunicación verbal se ve disminuida en ocasiones debido a la presencia de ambigüedades, las cuales, se refieren al hecho de que la información comunicada puede ser interpretada de más de una forma. La ambigüedad se genera a menudo por limitaciones de las personas al usar



un lenguaje, deficiencias propias de este o ambas. La omisión de información es también un factor que afecta la eficiencia de la comunicación verbal. Por ejemplo, durante la presentación el arquitecto podría omitir información relevante acerca de la arquitectura por considerarla obvia para su audiencia.

En este contexto afirmamos que la existencia de documentación adecuada podría minimizar todos estos problemas. El arquitecto podría utilizar la documentación de la arquitectura como un elemento de soporte durante la presentación verbal de esta a los diferentes involucrados en el sistema.

4.3.2 Preservar información sobre la arquitectura

La documentación es persistente por naturaleza. De esta forma, la existencia de documentación sobre la arquitectura, y el acceso a aquella misma, podría ser de utilidad para resolver varias situaciones que pueden presentarse durante el desarrollo de un sistema.

Por ejemplo, si nuevos miembros se unen al equipo de desarrollo, pueden usar la documentación para comprender de forma más rápida la arquitectura del sistema. Esto es positivo y adecuado, comparado con el escenario de depender de la disponibilidad del arquitecto para conocer de su propia voz toda la información al respecto.

De forma similar, la existencia de documentación permite que cuando personas clave involucradas en el diseño de la arquitectura se ausentan, el resto del equipo de desarrollo puede consultarla para obtener la información requerida sin tener que esperar a que estas personas regresen.

Otro ejemplo es que las actividades de mantenimiento o actualización a un sistema pueden llevarse a cabo de manera más localizada si existe un documento que describe la arquitectura de este. Este ejemplo es más valioso si pensamos en que el sistema en turno es uno legado, el cual fue diseñado y desarrollado por personal que ya no labora en la compañía.

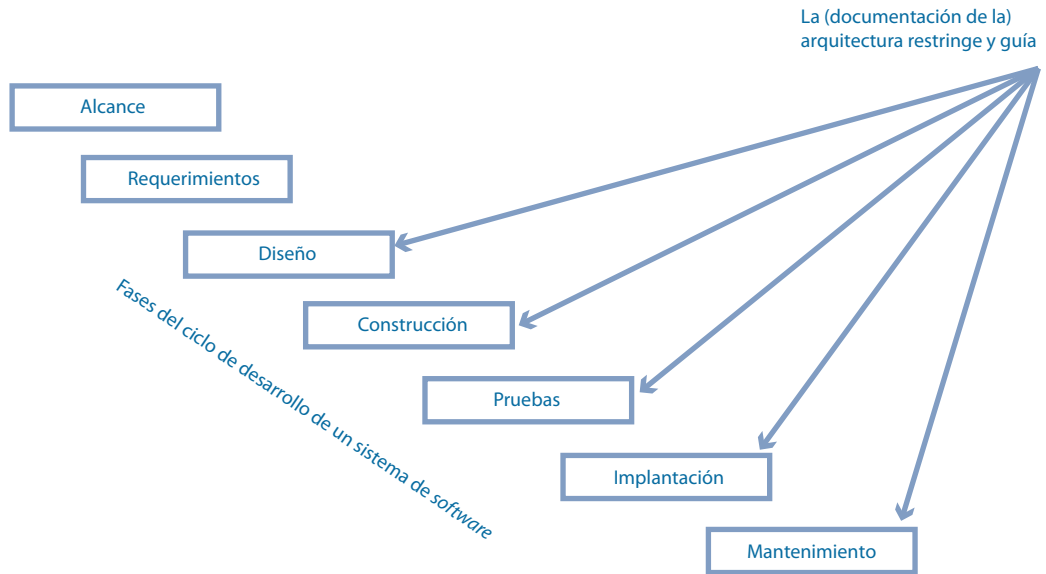
Todos los cambios realizados a la arquitectura deberían ser registrados en la documentación. De esta forma, estos pueden preservarse y ser conocidos por aquellos que tienen acceso a la documentación de la arquitectura.

4.3.3 Guiar la generación de artefactos para otras fases del desarrollo

En los capítulos anteriores expusimos que la *arquitectura de software* representa la estructura con la cual se proporcionará lo que el cliente o usuario espera del sistema. Por tal razón la arquitectura es un artefacto muy importante para guiar otras actividades dentro del ciclo de desarrollo de sistemas. Aunque todas ellas se llevan a cabo por personas que pertenecen a diferentes grupos de interesados en el sistema, por ejemplo, los líderes de proyecto o quienes desarrollan o prueban el sistema, su realización requiere que todas estas personas tengan una visión compartida y se desenvuelvan en un contexto de trabajo común atendiendo a esta. Como se ilustra en la figura 4-2, la documentación de la arquitectura es este contexto de trabajo común que permite guiar y restringir la forma en que se llevan a cabo varias actividades durante determinadas fases del ciclo de desarrollo del sistema.

Por ejemplo, al contener información acerca de todas las estructuras que conforman la arquitectura, la documentación arquitectónica sirve como fuente para las personas encargadas de la planeación del desarrollo del sistema. De forma similar, al contener información sobre el particionamiento y la funcionalidad que albergan los elementos que conforman las estructuras, así como de las relaciones de estos elementos en la arquitectura, la documentación arquitectónica es útil como guía para las actividades relacionadas con la construcción y la prueba del sistema.

En la sección 4.4 explicaremos que además de los de *software*, los elementos que conforman las estructuras arquitectónicas pueden ser también de *hardware*; por ejemplo los equipos de cómputo en donde se instalan los elementos de *software*. Para promover algún *driver*, durante el diseño el arquitecto pudo haber decidido colocar los equipos de cómputo en ubicaciones geográficamente dispersas. Al contener esta información, la documen-



► **Figura 4-2.** La arquitectura guía y restringe determinadas actividades durante algunas fases del ciclo de desarrollo del sistema.

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

tación arquitectónica permite guiar las actividades de las personas encargadas de la implantación o el mantenimiento de sistema.

Ya hemos mencionado que en métodos como el ACDM (Lattanze, 2008) la documentación de la arquitectura es un documento indispensable para llevar a cabo de manera efectiva las actividades de evaluación. En el capítulo 5 describiremos con mayor detalle la etapa de evaluación de la arquitectura y observaremos que la identificación de problemas depende casi totalmente de la existencia de una buena documentación de la arquitectura.

4.3.4 Proveer un lenguaje común entre diversos interesados en el sistema

La documentación describe las estructuras arquitectónicas con el propósito de que la información relacionada pueda ser comunicada eficientemente a los diversos interesados en el sistema. Alrededor de esta idea ya hemos señalado cómo el documentar la arquitectura ayuda a disminuir algunas limitaciones de la comunicación verbal, a minimizar en algunos casos la necesidad del usar comunicación verbal para describir la arquitectura, o bien, a guiar actividades diversas durante el desarrollo de un sistema. Sin embargo, es importante mencionar una característica muy importante que permite que estos aspectos positivos ocurran: la documentación de la arquitectura debe ser elaborada utilizando una notación o lenguaje que tenga sintaxis y semántica bien definida; esto es, una que al ser observada o utilizada por los diversos interesados en el sistema se interprete de la misma forma por todos ellos. Cuando la documentación de la arquitectura está elaborada usando una notación adecuada, esta notación y las estructuras representadas por medio suyo se convierten en un lenguaje común entre los interesados.

En la sección 4.5 proveemos más detalles sobre este punto al describir los diversos tipos de notaciones que se pueden utilizar para elaborar la documentación de la arquitectura.

4.4 VISTAS

Ya hemos explicado que la documentación de la arquitectura describe sus estructuras con el propósito que la información sobre estas pueda ser comunicada eficientemente a los diversos interesados en el sistema. Dichas estructuras se conforman por diversos elementos que, de acuerdo con la perspectiva desde la cual son descritos, adquieren una identidad particular. Por ejemplo, los elementos que adquieren la forma de unidades de implementación son utilizados por lo habitual cuando se hace una descripción sobre cómo está implementada la arquitectura. Sin embargo, estos mismos elementos pueden convertirse en una o más instancias al hacer una descripción sobre lo que ocurre cuando la arquitectura se ejecuta. De forma similar, cuando se elabora una descripción sobre cómo y dónde se encuentra instalada la arquitectura, las estructuras que la conforman pueden incluir elementos de *hardware* o incluso algunos que representan ubicaciones geográficas.

De esta forma, la heterogeneidad de las estructuras y sus elementos hace que resulte muy complicado describirlos a todos en un solo documento. Además, y debido a la diversidad de los interesados en el desarrollo de un sistema, no todos los elementos, estructuras y perspectivas son relevantes para cada uno de ellos. De esta forma, para facilitar el manejo de toda la información, un concepto ampliamente utilizado al elaborar la documentación es el de *vista*.

Una vista describe una o más estructuras de la arquitectura en términos de los elementos que la conforman.

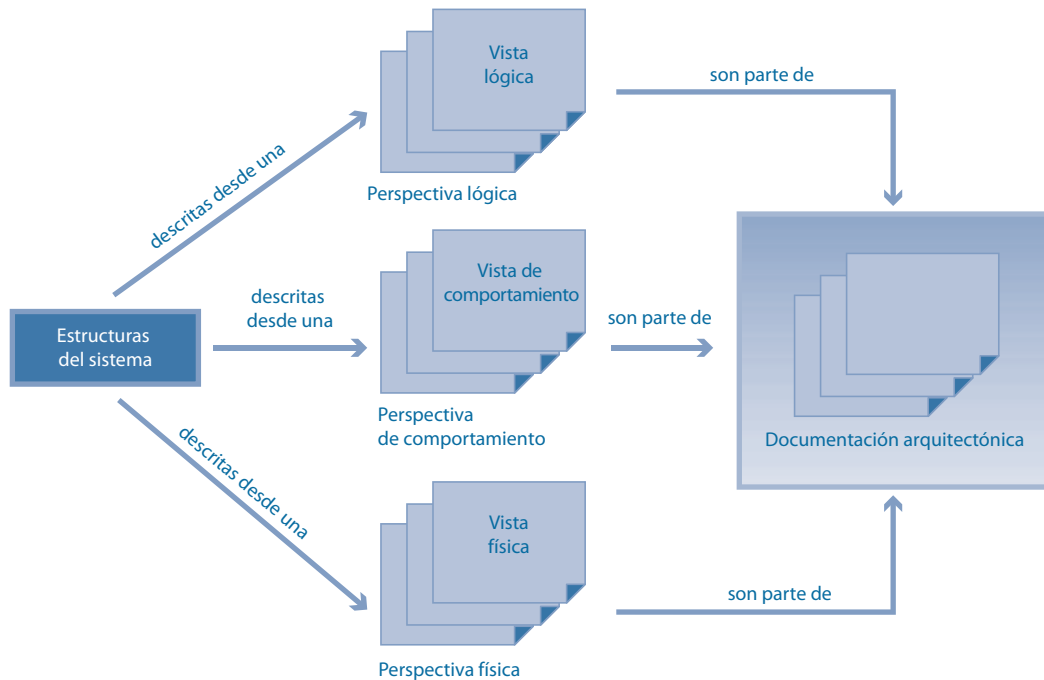
En términos generales una vista se conforma por: 1) un diagrama en donde se representan los elementos de la estructura, y 2) información textual que ayuda comprender este. Los diagramas permiten exhibir los elementos de la estructura de forma que es más fácil visualizarlos; la información textual describe por lo habitual las *propiedades y relaciones* definidas por elemento representado en el diagrama. Aunque en esta sección proveemos algunos ejemplos, en la sección 4.5 describiremos con mayor detalle las notaciones que pueden utilizarse para generar los diagramas y la información textual.

En el contexto de la etapa de documentación se reconocen varios tipos de vistas, por ejemplo, vistas de proceso, de desarrollo, de contexto o de implantación. De manera independiente al número total de vistas, es recomendable que el arquitecto documente al menos una de las siguientes tres clases:

1. Vistas lógicas.
2. Vistas de comportamiento.
3. Vistas físicas.

Como se observa en la figura 4-3, estas vistas toman sus nombres de las perspectivas más comunes desde las cuales un sistema puede ser descrito. Generar vistas dentro de estas tres clases es muy importante para comunicar eficientemente los aspectos clave de la arquitectura a los distintos interesados en el sistema. Si bien es cierto que cada vista describe aspectos específicos del sistema, es necesario mantener un mapeo entre la información que cada una de las vistas existentes contiene. De tal forma, usar el mismo nombre para referirse a un elemento de la arquitectura que aparece en varias vistas es una práctica indispensable. En la sección 4.7 se describen esta y otras recomendaciones relacionadas a la elaboración de vistas.

En las siguientes secciones describiremos algunas vistas tomando como ejemplo el caso referente al sistema de venta de boletos de autobús del que hemos hablado en el libro. Sin embargo, la sección 4 del apéndice contiene la documentación completa para el diseño presentado en el apéndice en la sección 3.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

› **Figura 4-3.** Perspectivas, vistas y documentación arquitectónica. Figura adaptada de (Lattanze, 2008).

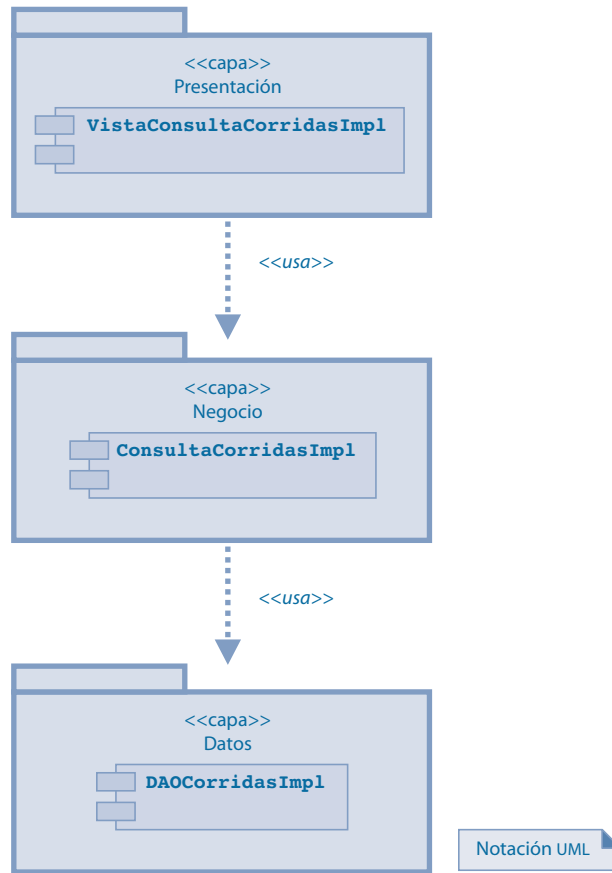
4.4.1 Vistas lógicas

Estas vistas describen las estructuras arquitectónicas en términos de elementos que toman la forma de unidades de implementación considerando tanto las propiedades como las relaciones u organización de cada una de estas. Las propiedades incluyen aspectos como, por ejemplo, su nombre, las funcionalidades o responsabilidades que le han sido asignadas, las interfaces que define o el lenguaje de programación utilizado para su implementación. La información sobre las relaciones u organización de las unidades se describen aspectos como, por ejemplo, qué funcionalidades o responsabilidades *ofrece a* o *espera* de otras unidades, o bien, cómo se agrupan en ensambles o jerarquías.

Algunas de las unidades de implementación que se utilizan frecuentemente en este tipo de vistas incluyen *clases*, *paquetes*, *módulos* o *subsistemas*. Asimismo, las relaciones o formas de organización utilizadas con frecuencia incluyen *es-parte-de* (que se emplea para representar una relación de agregación o composición), *depende-de* (la cual se usa para indicar que la unidad requiere de otra para funcionar correctamente), y *es-un* (que se utiliza para indicar que la unidad es “hija” de otra unidad la cual funge como “padre” y, por lo tanto, la unidad “hija” posee algunas de sus características).

La figura 4-4 muestra un ejemplo de vista lógica. Esta describe parte de la estructuración general del sistema de venta de boletos de autobús —usado como caso de estudio en este libro— en términos de capas que denotan distintas responsabilidades del sistema. La vista muestra también algunos de los módulos en estas capas, específicamente los que soportan la funcionalidad de consultar corridas. La relación `<<usa>>` se utiliza para indicar el tipo de vínculo entre los elementos de las vistas; `<<usa>>` es el mecanismo para indicar una relación *depende-de* en UML¹.

¹ Unified Modeling Language, por sus siglas en inglés.



› Figura 4-4. Ejemplo de una vista lógica.

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

4.4.2 Vistas de comportamiento

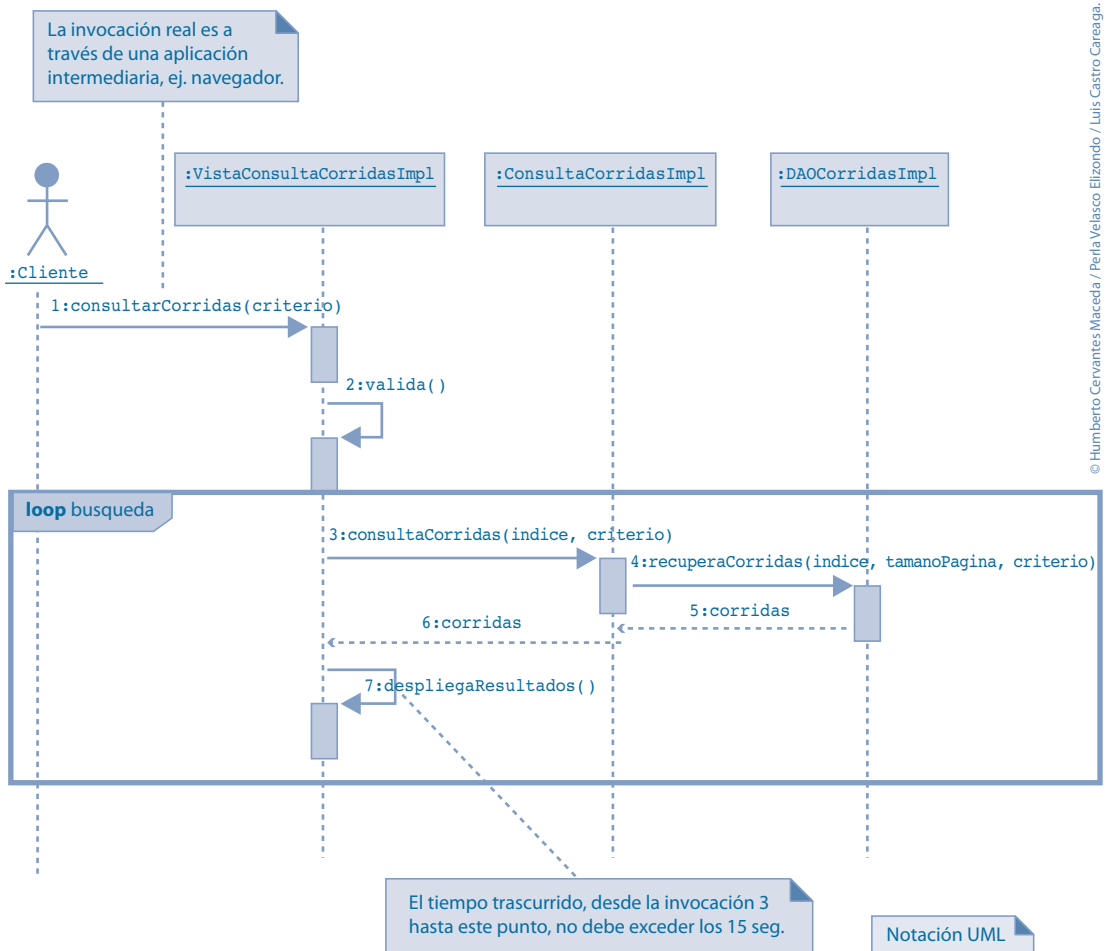
Las vistas de comportamiento describen estructuras cuyos elementos denotan entidades visibles en tiempo de ejecución, por ejemplo, *instancias*, *procesos*, *objetos*, *clientes*, *servidores* o *almacenes de datos*.

Además del nombre, el tipo y la funcionalidad ofrecidos, las propiedades de los elementos en este tipo de vistas incluyen por lo habitual información sobre valores específicos de atributos de calidad que pueden observarse cuando se ejecuta el sistema, por ejemplo, confiabilidad, desempeño o seguridad. Por otra parte, las relaciones describen la naturaleza de los mecanismos o protocolos de comunicación utilizados para la comunicación entre los elementos de la vista. De esta forma, *flujo-de-datos*, *llamada-a-procedimiento-remoto*, *acceso-compartido*, *broadcasting*, o incluso *SOAP*, son ejemplos válidos de relaciones que pueden emplearse durante la documentación de vistas de comportamiento (Clements, et al., 2010).

Es importante mencionar que en algunos enfoques de diseño y documentación, como en los descritos en Clements et al. (2010), las relaciones en este tipo de vistas se denotan como conectores.² Cuando este es el caso, el arquitecto puede definir y especificar propiedades para estos de la misma manera que se hace para elementos como procesos, clientes o servidores.

² En términos generales, un conector es un elemento arquitectónico que se usa para representar algún tipo de interacción entre componentes.

La figura 4-5 muestra un ejemplo de vista de comportamiento que describe aspectos relacionados con la funcionalidad para consultar corridas en nuestro ejemplo del sistema de venta de boletos de autobús. Como se puede observar, los elementos en esta vista son objetos, y las relaciones entre ellos son llamadas síncronas a procedimientos. Haciendo uso de comentarios se comunica que, mediante el uso de la táctica de paginación, el tiempo transcurrido desde que el cliente realiza una consulta hasta que se despliegan los resultados no debe exceder 15 segundos.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

> Figura 4-5. Ejemplo de una vista de comportamiento.

4.4.3 Vistas físicas

Estas vistas describen estructuras conformadas por elementos “físicos” que mantienen algún tipo de relación con los de las estructuras documentadas en otras vistas. Ya hemos mencionado que al documentar una arquitectura es relevante también saber, por ejemplo, cuáles equipos de cómputo albergan las unidades de implementación, cuáles ejecutan las instancias generadas a partir de ellas, y dónde se localizan físicamente estos. Cuando se documenta qué equipos las albergan, es posible que en uno solo estén más de una unidad de implementación. Por

esta razón es también importante documentar las localizaciones exactas de las unidades de implementación en las unidades de almacenamiento de los equipos de cómputo.

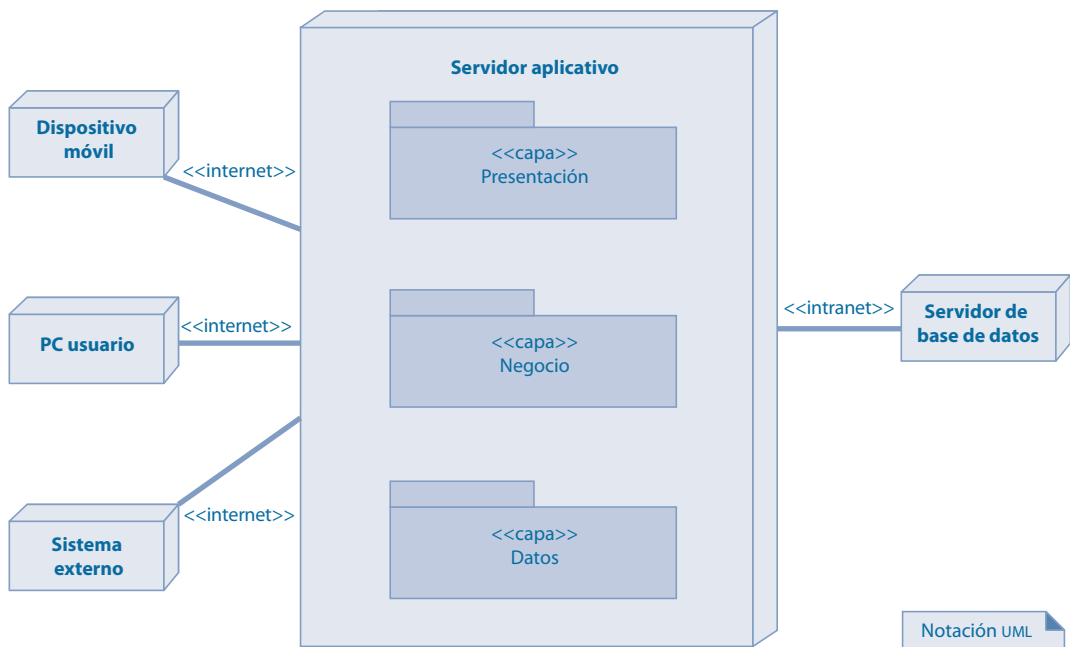
Con base en lo anterior podemos decir que en esta vista se pueden utilizar elementos de dos categorías: de *hardware* y de *software*.

Los equipos de cómputo y dispositivos como sensores o alarmas son quizá los elementos más comunes a considerar en la categoría elementos de *hardware*. Algunas de las propiedades que podrían considerarse al documentar algunos de ellos incluyen valores específicos acerca de información como modelo, marca, tipo de procesador, cantidad de memoria o capacidad de almacenamiento. En la mayoría de los casos, los valores que toman estas propiedades promueven la satisfacción de *drivers arquitectónicos*. Por ejemplo, un tiempo de respuesta de *x* segundos, requerido para la ejecución de determinado proceso, puede ser promovido mediante el uso de un equipo de cómputo con un procesador tipo *y*.

Respecto de los elementos de *software*, ejemplos válidos en estas vistas incluyen unidades de implementación como clases, paquetes, módulos o subsistemas, unidades visibles a tiempo de ejecución como instancias, procesos, objetos, clientes o servidores, o bien, almacenes de datos, así como estructuras de directorios. Para estos elementos se podrían o no definir propiedades. Cuando es el caso, estas denotan aspectos y valores relevantes en el contexto de implantación o producción del sistema, por ejemplo, permisos de acceso, sistema operativo, cantidad de memoria, almacenamiento o ancho de banda requerido.

Dada la naturaleza de estos elementos en las vistas físicas, relaciones como *reside-en* o *se-ejecuta-en*, son utilizadas con frecuencia.

La figura 4-6 ejemplifica una vista física en la cual se indica en qué equipos residen las capas resultantes de la estructuración general del sistema de ventas de boletos de autobús. La vista muestra también otros elementos físicos relevantes para el sistema y, haciendo uso de estereotipos en UML, la naturaleza de los canales de comunicación que permiten la interacción de todos estos elementos.



› Figura 4-6. Ejemplo de una vista física.

Es importante mencionar que las vistas físicas pueden emplearse también para documentar aspectos de asignación de trabajo, por ejemplo, para indicar elementos de las vistas lógicas a las personas del equipo de desarrollo que serán responsables de su implementación.

4.5 NOTACIONES

En la sección 4.3.4 mencionamos que un aspecto fundamental para que la documentación cumpla el objetivo de comunicar de manera eficiente información sobre las estructuras es el uso de notaciones con una sintaxis y semántica que minimice, o idealmente erradique, la posibilidad de ambigüedad en su interpretación. Se reconoce por ello que las notaciones formales funcionan mejor, pues usan un lenguaje preciso, esto es, uno con sintaxis y semántica bien definidas, por lo habitual basadas en algún fundamento matemático. Sin embargo, no siempre es necesario ni práctico emplear este tipo de notaciones, y por ello existen otras que podrían ser también adecuadas para documentar la *arquitectura de software*.

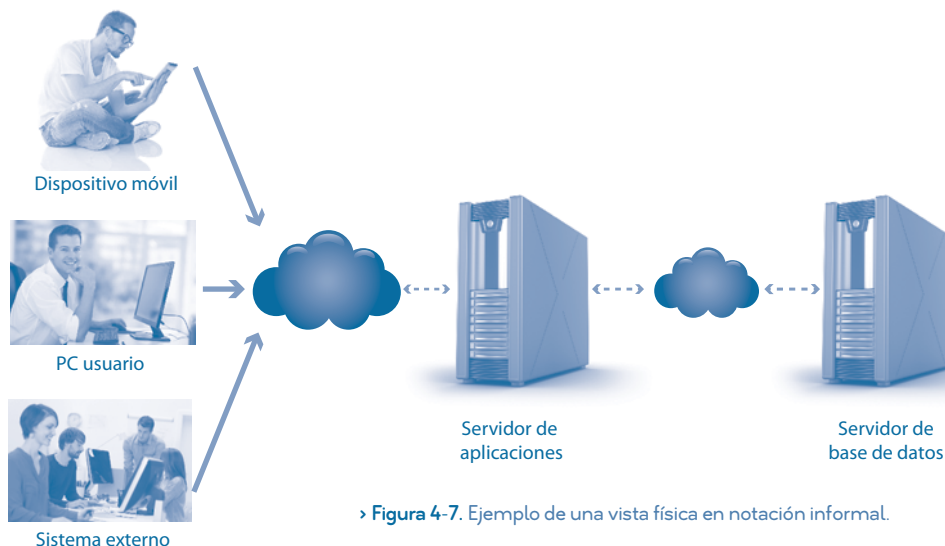
En las siguientes secciones describiremos algunas notaciones que, de acuerdo con nivel de formalidad, pueden clasificarse en:

1. Notaciones informales.
2. Notaciones semiformales.
3. Notaciones formales.

4.5.1 Notaciones informales

Como su nombre lo sugiere, estas notaciones son todas aquellas que no utilizan un lenguaje preciso. Esto es, ni su sintaxis ni su semántica son formales y, por lo tanto, pueden resultar inadecuadas si no se utilizan atendiendo determinadas recomendaciones. Por ejemplo, incluir siempre una descripción en lenguaje natural que explique el significado de cada elemento de la notación, utilizar los mismos elementos de la notación para representar lo mismo en todas las vistas. En la sección 4.7 hablaremos de otras recomendaciones para reducir la ambigüedad cuando se utiliza una notación informal para documentar la arquitectura.

La figura 4-7 muestra un ejemplo de vista física elaborada usando una notación informal. Como lo hemos mencionado, si no se incluye una descripción sobre el significado de cada elemento que aparece en la vista, su interpretación puede ser equivocada. Por ejemplo, no es claro por qué se usan diferentes tipos de líneas para



› Figura 4-7. Ejemplo de una vista física en notación informal.

conectar los elementos ni tampoco por qué unas son unidireccionales y otras bidireccionales. Tampoco es aclarado el significado de las dos nubes.

4.5.2 Notaciones semiformales

En sentido estricto, las notaciones semiformales tampoco están basadas en un lenguaje preciso. La noción de “semiformal” describe más bien la situación de que utilizan una sintaxis y semántica “aceptada ampliamente” dentro de un dominio de aplicación para la cual se tiene. Ejemplos de estas notaciones incluyen el lenguaje de modelado unificado (UML) o la notación entidad-relación.

Es importante mencionar que estas notaciones no han sido creadas de manera exclusiva para documentar *arquitecturas de software*, por lo que en algunos casos resultan limitadas para expresar aspectos específicos en este contexto. A partir de la versión 2 de UML se han realizado algunas adiciones para facilitar el modelado arquitectónico. Sin embargo, siguen siendo limitadas en particular para documentar vistas dinámicas. Por esta razón, en Clements *et al.* (2010) se proveen algunas guías para el uso de UML 2.x en la documentación de arquitectura.

Al igual que las notaciones informales, las semiformales deben ser utilizadas con cuidado para evitar ambigüedad en su interpretación.

Las figuras 4.4 a la 4.7, presentadas en secciones anteriores, muestran ejemplos de vistas de arquitectura en UML.

4.5.3 Notaciones formales

Como lo mencionamos antes, las notaciones formales utilizan un conjunto de conceptos arquitectónicos, los cuales se describen usando un lenguaje con fundamento matemático, como algún tipo de lógica o de álgebra. Tales conceptos en estos lenguajes son por lo habitual: componente –el cual representa un elemento que realiza procesamiento o almacenamiento de datos–; conector –con el que se hace representación de canales de flujo de datos o de control entre componentes–, y configuración –el cual representa la forma particular en la que los componentes y conectores se relacionan en el sistema que se describe.

A pesar de no ser muy utilizadas en la práctica, en la academia existen muchas notaciones formales de las cuales se puede hacer uso para documentar la arquitectura. La mayoría de ellas son reconocidas como lenguajes de descripción de arquitecturas, o ADL,³ por sus siglas en inglés. Al ser notaciones creadas con el fin de documentar arquitecturas, estas notaciones no presentan las limitaciones de las informales y semiformales. Además, debido a su tipo de fundamento matemático, estas notaciones pueden analizarse mediante el uso de métodos formales. Por ejemplo, al utilizar los valores provistos para propiedades, como tiempo de ejecución, periodicidad y prioridad de un conjunto de procesos, se podrían usar métodos de planificación de tareas para determinar si pueden ejecutarse de forma concurrente dentro de un lapso de tiempo determinado.

Algunos de los ADL más conocidos son *Acme*⁴, creado en la Universidad Carnegie Mellon; *AADL*⁵, creado tanto por miembros de esta misma universidad como el SEI, la Armada de los Estados Unidos y *Honeywell Technology Center*, y *xADL*⁶ creado en la Universidad de California Irvine.

La figura 4-8 muestra un pequeño extracto de la documentación de una vista lógica del sistema de nuestro caso de estudio usando el lenguaje de descripción de arquitectura *Acme*. La vista incluye información sobre dos componentes (*VistaConsultaCorridas* y *ConsultaCorridasImpl*) y un conector (*rmi*). Los componentes y conectores tienen puertos y roles, denotados con las palabras *Port* y *Role*, los cuales funcionan como interfaces que les permiten la interacción. La forma particular en la que los componentes y conectores se relacionan en este sistema, es decir, la configuración, está especificada en el bloque de información dentro de la palabra

³ *Architecture Description Language*.

⁴ <http://www.cs.cmu.edu/~acme/>

⁵ <http://www.aadl.info/>

⁶ <http://isr.uci.edu/projects/xarchuci/index.html>

Attachments. La documentación muestra también la definición de algunas propiedades, denotadas con la palabra *Property*, relevantes para los componentes descritos; por ejemplo *conexionesMax*, para especificar el número de conexiones concurrentes que puede soportar el componente, y *latenciaMax*, para especificar el tiempo máximo de procesamiento de una solicitud.

```
System RedADM = {  
    ...  
  
    Component VistaConsultaCorridas = {  
        Port p_envia;  
        Property codigoFuente: externalFile = "consulta.jsp";  
    };  
  
    Component ConsultaCorridasImpl = {  
        Port p_recibe;  
        Property conexionesMax: integer=100;  
        Property latenciaMax: integer= 2;  
        Property codigoFuente: externalFile = "consulta.java";  
    };  
  
    Connector rmi = {  
        Role r_cliente;  
        Role r_cliente  
    };  
  
    Attachments {  
        cliente.envia to rmi.r_cliente;  
        servidor.recibe to rmi.r_cliente;  
    }  
    ...  
}
```

› Figura 4-8. Ejemplo de vista en notación formal.

4.6 MÉTODOS Y MARCOS CONCEPTUALES DE DOCUMENTACIÓN DE ARQUITECTURA

Hasta este momento nos hemos limitado a describir la etapa de documentación en términos de elaborar vistas utilizando algún tipo de notación. Sin embargo, poco hemos dicho sobre cómo decidir de manera más sistemática cuáles vistas y de qué modo elaborarlas. En el soporte existente en este rubro es posible distinguir entre métodos y marcos conceptuales para la documentación de arquitecturas. Mientras los métodos describen de manera más explícita tanto las entradas requeridas como la secuencia de acciones que comprenden y las salidas generadas, los marcos conceptuales proveen un conjunto de conceptos que deben considerarse al documentar la arquitectura. Los métodos y marcos que presentamos abajo existen de forma independiente o están incrustados en un método específico de desarrollo de arquitectura.

En este capítulo describimos los siguientes métodos o marcos conceptuales:

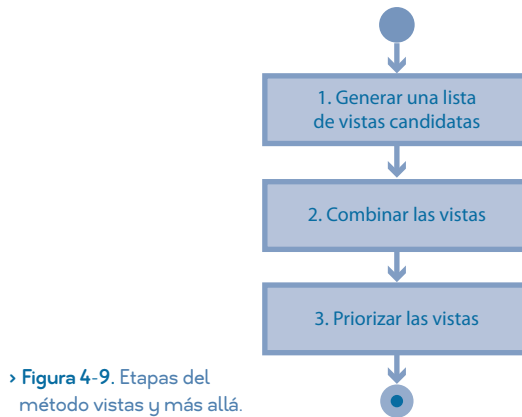
1. Vistas y más allá (*Views and Beyond*).
2. 4+1 Vistas (*4+1 Views*).
3. Método de diseño centrado en la arquitectura (ACDM).
4. Puntos de vista y perspectivas (*Viewpoints and Perspectives*).

4.6.1 Vistas y más allá

Vistas y más allá (Clements et al., 2010), o *Views and Beyond*, como expresa su nombre en inglés, es un método propuesto por el SEI para soportar la documentación de la *arquitectura de software* de un sistema. Considera tres categorías de vistas arquitectónicas relevantes, recomendaciones sobre qué documentar de cada una de ellas, así como una considerable cantidad de información adicional que puede complementar la documentación de la arquitectura.

Las categorías de vistas consideradas en Vistas y más allá son: vistas de módulos, vistas de componentes y conectores (C&C⁷), y vistas de asignación. En términos generales, todas estas son análogas a las vistas lógicas, de comportamiento y físicas descritas en la sección 4.4. La información adicional considerada en este método corresponde a la del sistema, la cual es relevante para todas las vistas, por ejemplo, descripción del sistema, relación entre vistas, glosarios, restricciones de negocio y *drivers arquitectónicos*.

Para apoyar al arquitecto en la tarea de documentar el diseño de la arquitectura, Vistas y más allá define un proceso secuencial simple en términos de las etapas que se ilustran en la figura 4-9 y se describen a continuación:



› Figura 4-9. Etapas del método vistas y más allá.

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Caneaga.

1. **Generar una lista de vistas candidatas.** Utilizando una tabla que en sus filas incluya el conjunto de interesados en el sistema, y en sus columnas, los diferentes tipos de vistas, el arquitecto identifica las vistas relevantes para el diseño en turno. De acuerdo con cada vista, el arquitecto debe indicar el nivel de detalle con que estas requieren ser documentadas.
2. **Combinar las vistas.** A efecto de reducir el número de vistas a documentar, el arquitecto lleva a cabo en esta etapa un análisis acerca de la tabla generada en el paso anterior. Podría detectar casos en los que, por ejemplo, algunas vistas requieren ser documentadas de forma muy general y que las necesitan muy pocas personas. Podría también detectar que existen vistas utilizadas para satisfacer los requerimientos de información de más de un interesado en el sistema, y que la combinación de ellas en una sola no implica mucha complejidad sino, por el contrario, ayuda a satisfacer los de todos estos interesados.
3. **Priorizar las vistas.** Este paso se lleva a cabo para decidir cuáles vistas, de la lista final, deben elaborarse primero. La priorización se realiza considerando los requerimientos de información para continuar con el diseño detallado del sistema o su implementación.

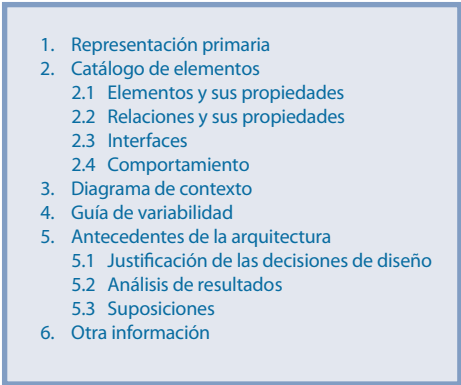
⁷ Component-and-Connector Views.

Vistas y más allá no hace especificación explícita del tipo de notación a utilizar, pero sí énfasis especial en hacer uso de una que minimice la ambigüedad en su interpretación. Para elaborar las vistas en cada una de las categorías, el método sugiere utilizar un conjunto de estilos arquitectónicos; por ejemplo, descomposición, uso o capas para las vistas de módulos. Se recomienda consultar Clements *et al.* (2010) para mayores detalles sobre los estilos en cada categoría de vistas.

El método también sugiere utilizar una plantilla, con siete partes fundamentales, para organizar las vistas y otra información generadas. La estructura de la plantilla se ilustra en la figura 4-10. Como puede observarse, la estructura de la plantilla incluye:

1. **Representación primaria de la vista.** Es una representación gráfica de la vista en términos de sus principales elementos y relaciones, así como el texto correspondiente para describir de forma general la información anterior.
2. **Catálogo de elementos.** Esta sección hace las veces de un catálogo que contiene los elementos de la representación primaria describiéndolos con detalle. La información explícita acerca de estos incluye, propiedades, interfaces o comportamiento.
3. **Diagrama de contexto.** Permite ubicar al lector de la documentación sobre cuál parte del sistema es la que se describe en la vista.
4. **Guía de variabilidad.** Describe los posibles puntos de variación permisibles, si es que existen, de los elementos y relaciones representados en la vista. Ejemplos de estas variaciones incluyen rangos en los valores de las propiedades de los componentes, o variaciones en los protocolos de comunicación soportados por ellos.
5. **Antecedentes de la arquitectura.** Explica las razones que soportan las decisiones de diseño tomadas por el arquitecto durante el diseño de elementos y relaciones representadas en la vista.
6. **Otra información.** En esta sección se describe información relevante extra que aplique a la vista. Algunos ejemplos de ella: reglas de negocio, información sobre el equipo de desarrollo o descripción del sistema.

En el contexto del caso de estudio del libro, en la sección 4.4 del apéndice se puede encontrar un ejemplo de documentación de vista, con el uso del método de Vistas y más allá.

- 
1. Representación primaria
 2. Catálogo de elementos
 - 2.1 Elementos y sus propiedades
 - 2.2 Relaciones y sus propiedades
 - 2.3 Interfaces
 - 2.4 Comportamiento
 3. Diagrama de contexto
 4. Guía de variabilidad
 5. Antecedentes de la arquitectura
 - 5.1 Justificación de las decisiones de diseño
 - 5.2 Análisis de resultados
 - 5.3 Suposiciones
 6. Otra información

› **Figura 4-10.** Plantilla de documentación sugerida en Vistas y más allá.

4.6.2 4+1 Vistas

4+1 Vistas es un marco conceptual para la documentación de arquitectura propuesto por Philippe Kruchten (Kruchten, 1995). Como su nombre lo indica, este marco considera la noción de vista como concepto principal; recomienda de modo específico la elaboración cinco vistas:

1. **Vista lógica.** Describe la arquitectura desde la perspectiva de los usuarios finales, es decir, considerando los elementos principales que proveen a estos los servicios del sistema.
2. **Vista de procesos.** Explica la arquitectura haciendo mayor énfasis en los elementos que en tiempo de ejecución permiten soportar aspectos como concurrencia, rendimiento o escalabilidad.
3. **Vista de desarrollo.** Describe la arquitectura en términos de elementos relevantes para los desarrolladores del sistema.
4. **Vista física.** Explica los componentes físicos del sistema (es análoga a las vistas físicas descritas en la sección 4.4.3)
5. **Vista “+1”** tiene la función de relacionar los elementos en las otras cuatro vistas por medio de casos de uso o escenarios que ilustren cómo interactúan todos estos elementos.

Cada una de estas vistas puede ser elaborada usando una plantilla, y es posible representar sus estructuras usando algún estilo o patrón arquitectónico.

Aunque 4+1 Vistas no especifica explícitamente el tipo de notación a utilizar, sí sugiere organizar la documentación arquitectónica en dos documentos. El primero, llamado Documento de arquitectura, contiene principalmente las vistas anteriormente descritas. Su estructura está representada en la figura 4-11. El segundo documento, denominado Guía sobre el diseño de la arquitectura, contiene información para entender el porqué de las decisiones de diseño tomadas por el arquitecto.

<p> Título Historial de cambios Tabla de contenido Índice de figuras </p> <ol style="list-style-type: none"> 1. Alcance 2. Referencias 3. <i>Arquitectura de software</i> 4. Objetivos de la arquitectura y restricciones 5. Arquitectura lógica 6. Arquitectura de procesos 7. Arquitectura de desarrollo 8. Arquitectura física 9. Escenarios 10. Tamaño y desempeño <p> Apéndices A. Acrónimos y abreviaciones B. Definiciones C. Principios de Diseño </p>
--

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Carreaga.

› Figura 4-11. Estructura del Documento de arquitectura sugerida en 4+1 Vistas.

4.6.3 Puntos de vista y perspectivas

Como explicamos en el capítulo anterior, Puntos de vista y perspectivas (Rozanski y Woods, 2005) es un método de diseño de arquitectura. De esta forma, lo que describiremos en esta sección es el enfoque de documentación que este método considera.

Recordando, Puntos de vista y perspectivas plantea que la arquitectura debe estructurarse considerando un conjunto de vistas, las cuales tienen que ser elaboradas tomando en cuenta la información definida por *un punto de vista*. De manera adicional, el método hace hincapié en la noción de *perspectiva arquitectónica* que denota un conjunto de actividades, tácticas, riesgos y aspectos de importancia para enriquecer una vista respecto de cómo esta soporta un conjunto de atributos de calidad.

Cada punto de vista está definido en términos de una descripción, aspectos de interés (por ejemplo sincronización o latencia), modelos notacionales a utilizar (por ejemplo máquinas de estado, de flujo de datos o diagramas), posibles problemas durante su elaboración (por ejemplo abrazo mortal o ambigüedad), audiencia (por ejemplo desarrolladores o administradores del sistema) y aplicabilidad.

El método define siete puntos de vista, y estos indican a su vez el tipo y el número de vistas a documentar:

1. **Funcional:** referente a vistas que describen elementos funcionales del sistema.
2. **Información:** hace alusión a vistas que explican cómo los elementos almacenan, manipulan, administran y distribuyen la información del sistema.
3. **Concurrencia:** hace referencia a vistas que describen la manera en que los elementos de la arquitectura soportan aspectos relacionados con la concurrencia.
4. **Desarrollo:** referente a vistas que especifican cómo los elementos deben ser implementados, probados, o modificados.
5. **Implantación:** alude a vistas que señalan cómo los elementos deben ser implantados en su ambiente de producción.
6. **Operacional:** hace referencia a vistas que indican el modo en que los elementos deben utilizarse en su ambiente de producción.
7. **Contexto:** alude a vistas que describen las dependencias, relaciones e interacciones de los elementos de la arquitectura y su ambiente.

En contraste con los métodos anteriores, este no sugiere de manera explícita alguna estructura para organizar la documentación generada. Sin embargo, hay determinado énfasis respecto del uso en UML como notación preferida.

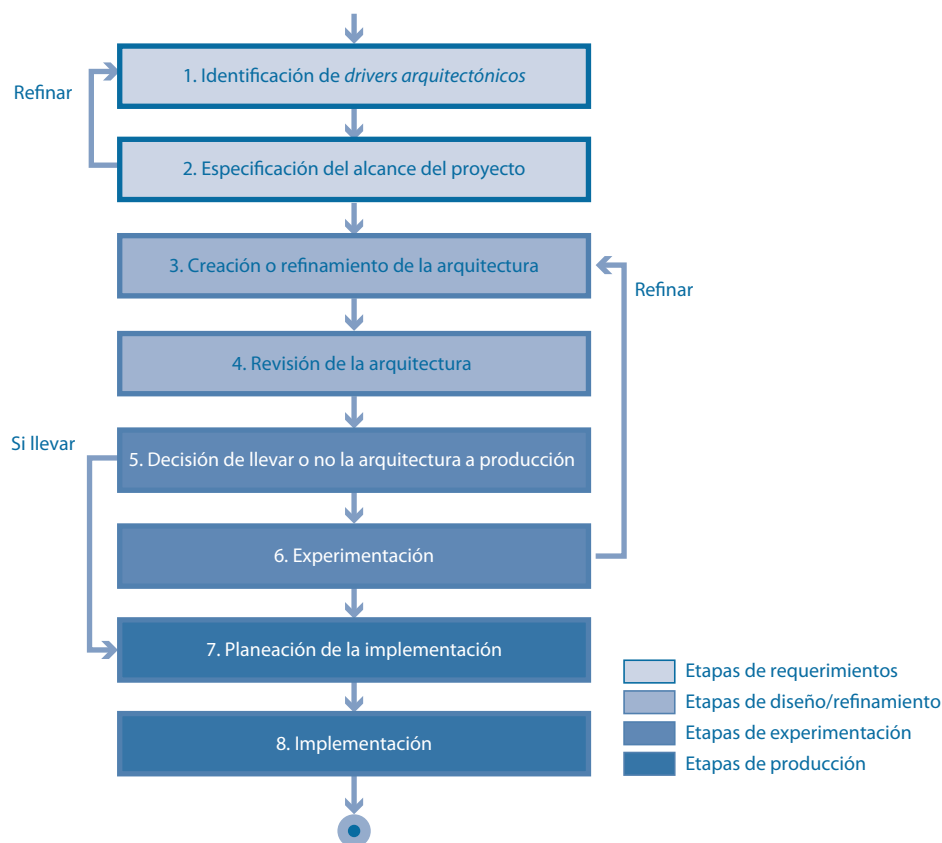
4.6.4 ACDM (etapas 3 y 4)

ACDM (Lattanze, 2008) es un método de diseño de *arquitectura de software*; en los capítulos 2 y 3 hemos descrito partes de él. Durante la etapa 3, Creación o refinamiento de la arquitectura, se define el diseño inicial o se refina la versión actual de este en relación con los resultados obtenidos en la etapa 4, Revisión de la arquitectura (véase la figura 4-12).

Ya mencionamos también que, en el ACDM, el diseño y la documentación no son etapas mutuamente excluyentes. Así, una parte de las actividades de la etapa 3 incluye la creación o refinamiento de la arquitectura y la creación o actualización de la documentación correspondiente. Estas actividades consideran la participación de los siguientes roles: administrador del proyecto, arquitecto de *software* líder, líder científico, ingeniero de requerimientos, ingeniero de calidad, ingeniero de soporte e ingeniero de producción.

ACDM considera las vistas estáticas, dinámicas y físicas que, en términos generales, son análogas a las lógicas, de comportamiento y físicas, descritas en la sección 4.4. Contempla además la noción de vistas mixtas utilizadas para documentar información necesaria para, por ejemplo, comunicar la asignación de elementos de vista dinámica a elementos de la vista física o a los equipos responsables de su implementación, o bien, comunicar la relación entre los *drivers arquitectónicos* y los elementos que los soportan en las diferentes vistas.

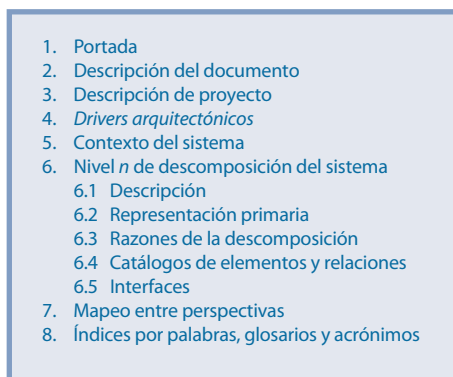
Para la selección de las vistas, el ACDM sugiere adoptar una estrategia similar a la de Vistas y más allá. Esto es, identificar a los interesados en el sistema y, considerando los tipos de vistas, identificar para cada uno de estos las vistas que les son relevantes, así como el nivel de detalle con que requieren ser documentadas.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

› Figura 4-12. Etapas del ACDM en donde se elabora la documentación de la arquitectura.

Cabe señalar que el ACDM sugiere organizar la documentación de acuerdo con la estructura presentada en la figura 4-13. ACDM no hace énfasis en el uso de una notación específica.



› Figura 4-13. Estructura del Documento de arquitectura sugerida en el ACDM.

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

4.6.5 Otros

Además de los métodos y marcos conceptuales descritos anteriormente, existen algunos otros recursos que el arquitecto puede utilizar durante la documentación de la arquitectura. A continuación los describimos de modo breve.

El estándar ISO/IEC/IEEE 42010:2011, Descripción de arquitecturas, provee un marco conceptual para describir arquitecturas. Da también una especificación de lo que debe incluir un documento para considerarlo adecuado a este estándar. Entre sus conceptos relevantes para este capítulo hemos de mencionar los de vista y puntos de vista. El marco conceptual de este estándar es utilizado en el método Puntos de vista y perspectivas, descrito en la sección 4.6.3.

El modelo 4 Vistas de Siemens (Hofmeister, Nord y Soni, 1999) se crea a partir de un estudio realizado sobre la práctica de actividades de *arquitectura del software* en la industria. Establece que la documentación de la arquitectura debe llevarse a cabo considerando cuatro vistas principales: conceptual, de módulos, de ejecución y de código.

Finalmente, *Rational ADS* es una adaptación del 4+1 Vistas, creada para soportar la documentación de arquitecturas complejas, como las empresariales, de *e-business* o de sistemas incrustados. Además de las cinco vistas consideradas en 4+1 Vistas, las cuales son renombradas, *Rational ADS* considera cuatro vistas adicionales. Las nueve vistas resultantes (requerimientos no funcionales, casos de uso, dominio, experiencia del usuario, lógica, procesos, implementación, implantación y prueba) se agrupan de acuerdo con cuatro puntos de vista: requerimientos, diseño, realización y verificación.

4.6.6 Comparación de los métodos y marcos conceptuales

Presentamos la siguiente tabla comparativa con el propósito de proveer al lector un resumen de las características de los métodos y marcos conceptuales descritos en las secciones anteriores.

	Vistas y más allá	4+1 Vistas	Puntos de vista y perspectivas	Método de diseño centrado en la arquitectura (ACDM)
Tipo	Método	Modelo	Marco conceptual	Método
Mecánica y enfoque	Definido como una secuencia de pasos que soporta la identificación de las vistas a documentar.	Puede ser utilizado dentro del contexto de un método de desarrollo para soportar la especificación y documentación de vistas arquitectónicas.	Modelo que puede ser utilizado en el contexto de un método de desarrollo para soportar la especificación y documentación de vistas arquitectónicas.	Método iterativo e incremental que soporta la identificación y documentación de vistas arquitectónicas.
Provee un proceso que especifica las actividades, roles y artefactos de entrada y salida	Sí	No aplica	No aplica	Sí
Participantes	Arquitecto de <i>software</i> .	No aplica	No aplica	Se asume la participación de los siguientes roles: administrador del proyecto, arquitecto de <i>software</i> líder, líder científico, ingeniero de requerimientos, ingeniero de calidad, ingeniero de soporte e ingeniero de producción.

Continúa...

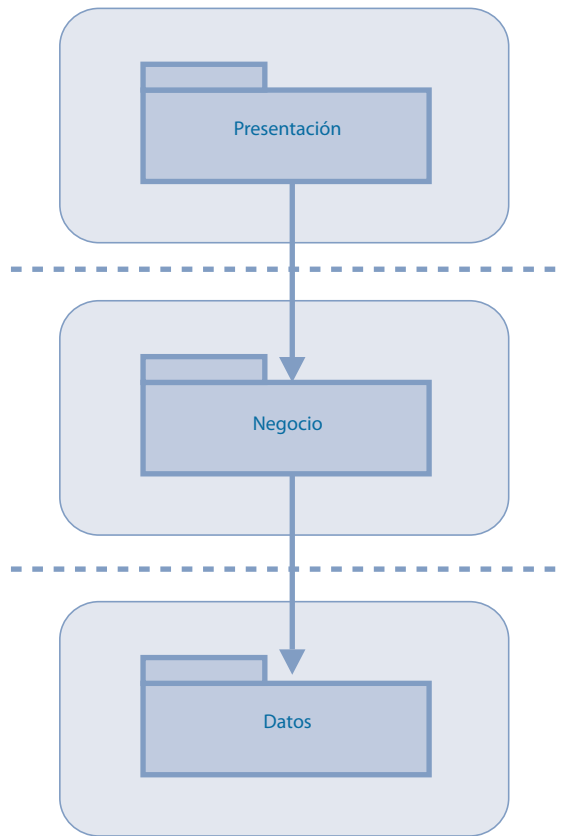
Continuación...

	Vistas y más allá	4+1 Vistas	Puntos de vista y perspectivas	Método de diseño centrado en la arquitectura (ACDM)
Tipo	Método	Modelo	Marco conceptual	Método
Entradas	Conjunto de interesados en el sistema e información proveniente de la etapa de diseño de la arquitectura.	No aplica	No aplica	<i>Drivers</i> de la arquitectura.
Salidas	Lista de vistas a documentar junto con el orden en que deben ser elaboradas.	No aplica	No aplica	Conjunto de vistas arquitectónicas documentadas.
Criterios de terminación	Tener un conjunto de vistas a documentar y el orden en que deben ser elaboradas.	No aplica	No aplica	Todas las vistas arquitectónicas están documentadas.
Considera las tres vistas fundamentales: lógicas, de comportamiento y físicas	Sí	Sí	Sí	Sí
Considera vistas adicionales	No	Sí	Sí	No
Considera información adicional del sistema que complementa la de las vistas (por ejemplo descripción del sistema, relación entre vistas, glosarios, restricciones de negocio y <i>drivers</i> arquitectónicos)	Sí	Sí	Sí	Sí
Provee soporte para documentar atributos no funcionales específicos	No	No	Sí	No
Da soporte para identificar o priorizar las vistas a documentar	Sí	No	No	Sí
Sigue cómo organizar la documentación generada	Sí	Sí	No	Sí
Notación utilizada	No indica	No indica	Hace determinado énfasis en el uso en UML.	No indica

4.7 RECOMENDACIONES PARA ELABORAR LA DOCUMENTACIÓN

En la sección anterior describimos algunos métodos y marcos conceptuales útiles durante la documentación de la arquitectura. Un aspecto común en estos métodos es que se enfocan en el “qué” y ofrecen poca orientación sobre el “cómo”. Esto es, se centran en indicar qué vistas se debe documentar y qué información de cada vista verter en documentos, pero ninguno explica con detalle cómo documentar adecuadamente las vistas.

Si bien, la selección de notaciones de documentación puede ofrecer determinado soporte en este sentido, a veces no resulta suficiente. Por ejemplo, la vista de la figura 4-14 se define usando elementos de notación de UML, pero sin la presencia del arquitecto difícilmente puede utilizarse para comunicar de modo eficiente la información que contiene. Algunas preguntas que surgen al observar esta vista son: ¿qué tipo de elementos representan los paquetes?, ¿qué clase de relaciones denotan las flechas que conectan los paquetes?, ¿qué elementos representan los cuadros debajo de los paquetes?, ¿qué significan las líneas punteadas horizontales?



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

› Figura 4-14. Ejemplo de vista de mala calidad.

A efecto de evitar este y otro tipo de problemas, durante la etapa de documentación se recomienda atender los aspectos siguientes e incrementar así la calidad de la documentación:

1. Escribir la documentación desde la perspectiva de la persona que la va a utilizar. El tipo de lenguaje usado en los documentos requeridos por los desarrolladores del sistema no es el más adecuado para comunicar la arquitectura al propietario de este.
2. En caso de usar notaciones propietarias o informales, incluir un cuadro de notación en cada diagrama para evitar ambigüedad en su interpretación.
3. Usar un nivel adecuado de abstracción o detalle considerando el tipo usuario de la documentación. El tipo de información que necesitan los desarrolladores no es el mismo que el requerido por quienes instalarán el sistema una vez desarrollado.



4. Cuidar aspectos de presentación como gramática, ortografía y legibilidad de las representaciones gráficas.
5. Minimizar el uso de acrónimos.
6. Usar nombres descriptivos para los elementos, relaciones y propiedades usados en una vista.
7. Maximizar la consistencia de información entre elementos que aparecen en las diferentes vistas y los otros artefactos generados.
8. Capturar siempre el porqué de las decisiones de diseño relevantes en cada vista.
9. Realizar una evaluación de la documentación antes de hacerla disponible.
10. Actualizar de manera periódica la documentación.

EN RESUMEN

En este capítulo abordamos la tercera etapa del ciclo de desarrollo de la arquitectura: la documentación. Su objetivo es generar la descripción de los elementos que conforman las estructuras arquitectónicas para que esta pueda ser comunicada de manera eficiente a los diversos interesados en el sistema.

Explicamos que para facilitar tal descripción, un concepto utilizado ampliamente durante la elaboración de los documentos de la arquitectura es el de vista. Los tipos de vista que se emplean con frecuencia son:

1. Vistas lógicas.
2. Vistas de comportamiento.
3. Vistas físicas.

PREGUNTAS PARA ANÁLISIS

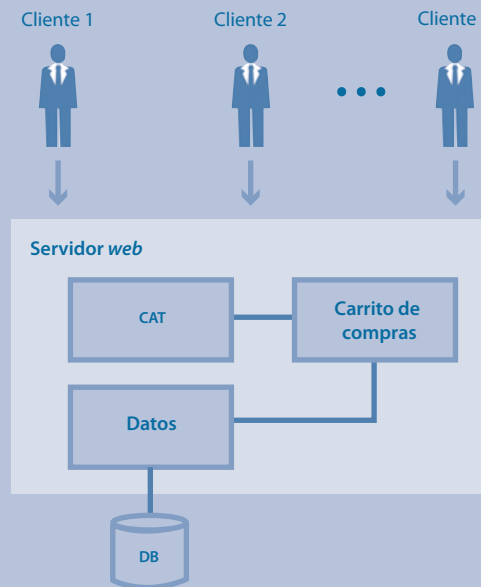
1. Hablando de documentación de arquitectura se reconocen varios tipos de vistas. Discuta con un(a) compañero(a) por qué se recomienda documentar siempre vistas lógicas, de comportamiento y físicas.
2. Discuta con un(a) compañero(a) las ventajas y desventajas de las notaciones informales respecto de las semiformales.
3. Discuta con un(a) compañero(a) en qué contextos son de mayor utilidad las notaciones formales de documentación de arquitectura.
4. ¿Considera que la arquitectura de un sistema puede ser documentada por completo usando una notación formal? Explique su respuesta.
5. En el contexto de ADL, ¿cuál es la diferencia entre una interfaz y un puerto?
6. Considere que una compañía de desarrollo de sistemas de *software* va a desarrollar un sistema de análisis automático de comentarios escritos en las redes sociales por los clientes de una cadena de restaurantes. Proponga una vista física para este sistema usando notación informal y discuta el porqué de los elementos, relaciones y propiedades en la vista.
7. Evalúe la calidad de la vista generada en la pregunta anterior en relación con algunos de los aspectos listados en la sección 4.7. En específico los aspectos 2 a 6.

Continúa...

Continuación...

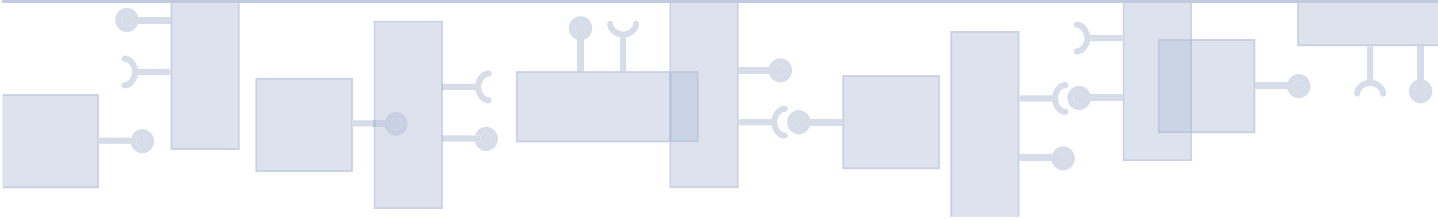
PREGUNTAS PARA ANÁLISIS

8. Considere que una compañía de sistemas va a desarrollar un sistema para una tienda de vinos, el cual permitirá a los clientes comprar utilizando un navegador *web*. Suponga que el arquitecto utiliza la vista mostrada a continuación para describir al equipo de desarrollo la arquitectura de este sistema. En su explicación, él menciona que "... el sistema se ejecuta en la parte superior de una base de datos denotada por DB, y además contiene un modelo de dominio denominado Datos, un catálogo de vinos denotado por CAT, y un carrito de compras que permite registrar las botellas que un cliente quiere adquirir". Discuta con un(a) compañero(a): ¿qué tipo de vista es esta? ¿Le parece que la vista es adecuada?



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.
Ilustración: © PureSolution / Shutterstock.

9. Evalúe la calidad de la vista presentada en la pregunta anterior respecto de algunos aspectos listados en la sección 4.7. En específico los aspectos 1 a 6.
10. Considere el sistema descrito en la pregunta 8. Suponga que este puede ser instalado y ejecutado en diferentes plataformas de *software* y *hardware*. ¿Cómo se podría representar ello en una vista?

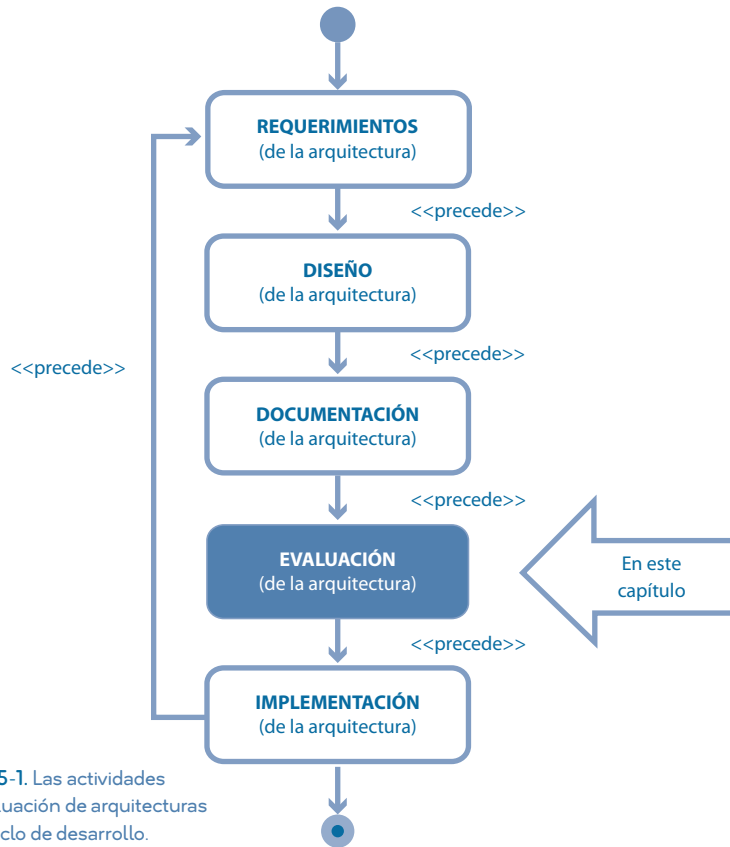


EVALUACIÓN: ASEGURAR LA CALIDAD EN LA ARQUITECTURA

En los capítulos anteriores planteamos formas de especificar, diseñar y documentar *arquitecturas de software*. Una vez que se cuenta con una arquitectura especificada, diseñada y documentada, es posible iniciar la construcción del diseño del sistema a partir de la misma. Sin embargo, dada la importancia de las decisiones que incorpora, es conveniente asegurarse de que sea correcta y satisfaga además los *drivers arquitectónicos*.

La evaluación de arquitecturas permite la detección de incumplimientos y riesgos asociados, que pueden ser causas de retrasos o problemas muy serios en los proyectos a materializar. Este capítulo se enfoca en la etapa de evaluación del ciclo de desarrollo de la arquitectura, como muestra la figura 5-1.

Iniciaremos planteando de forma general el concepto de evaluación, para después enfocarnos en la práctica evaluativa de la arquitectura y los principios que ella conlleva. Terminaremos el capítulo con la descripción de algunos métodos usados en la actualidad para realizar las evaluaciones.



› **Figura 5-1.** Las actividades de evaluación de arquitecturas en el ciclo de desarrollo.

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

5.1 CONCEPTOS DE EVALUACIÓN

Molestia y frustración son las reacciones que se tienen cuando los sistemas fallan o no hacen lo que el cliente espera. La evaluación es una técnica con la que los desarrolladores cuentan para evitar que las fallas lleguen a los usuarios finales o se presenten en momentos en los que corregirlas es complicado y costoso. En la evaluación se examina un producto o subproducto para ver si cumple con criterios de calidad y qué tanto se desvía de ellos.

Las desviaciones se clasifican en dos tipos: desviaciones de las necesidades reales de los usuarios del producto, y desviaciones a la construcción correcta del producto. Las evaluaciones que buscan la detección de las primeras se conocen como *validaciones*, y se hacen por lo habitual sobre los productos terminados, aunque también pueden practicarse tanto a documentos de requerimientos o de diseño como a cualquier otro artefacto de trabajo. En las validaciones participan todos los interesados en el uso del producto.

Las evaluaciones que buscan desviaciones de la construcción correcta se conocen como *verificaciones* y se hacen por lo habitual sobre los artefactos intermedios que los proyectos generan al ir concretándose. En ellas participan casi de manera exclusiva los mismos miembros de los equipos de desarrollo.

Las verificaciones y validaciones se consideran como unas de las mejores prácticas en la *ingeniería de software* para la detección temprana de errores, defectos y/o riesgos en los distintos artefactos o partes de los sistemas en desarrollo. En un nivel más específico, tanto las revisiones e inspecciones, como la elaboración de prototipos o experimentos, implementan muchos de sus objetivos. En un nivel más detallado y para el contexto de las *arquitecturas de software*, las inspecciones se especializan en evaluaciones de arquitectura.

5.2 EVALUACIÓN DE ARQUITECTURAS

La arquitectura es la base para el diseño y la construcción de un producto de *software*. Por lo mismo, es importante asegurarse de que no se desvíe de manera significativa de los *drivers arquitectónicos* (requerimientos funcionales, de atributos de calidad y de restricciones), así como tener confirmado que sea técnicamente correcta para que no se diseñen o construyan artefactos basados en arquitecturas erróneas o incompletas (Clements *et al.*, 2008).

La evaluación consiste en revisar, inspeccionar o recorrer los artefactos de la *arquitectura de software* buscando inconsistencias, errores o desviaciones respecto de los *drivers*. Sin ella, la implementación del sistema incurre en riesgos, los cuales, en caso de que se concreten, tienen costos e impactos en el proyecto que pueden ser desproporcionados. Por ello, una estrategia inteligente es incluir una etapa de control de calidad de los artefactos de requerimientos y diseño de las *arquitecturas de software*.

La evaluación de las arquitecturas permite conocer si los requerimientos obtenidos fueron correctos y completos, o bien, si han cambiado, y qué tan satisfactorias fueron las decisiones tomadas durante el diseño de la arquitectura. Se aplica principalmente a los artefactos generados en las etapas de requerimientos y diseño de arquitectura que ya se documentaron de manera idónea, aunque puede hacerse durante su realización o cuando el sistema ya está terminado.

5.3 PRINCIPIOS DE LA EVALUACIÓN

Las evaluaciones tienen como principio ser procedimientos que detecten lo antes posible los errores para que no afecten fases posteriores o el uso del producto. Buscan además que los requerimientos de este (final o intermedio) sean satisfechos, señalando los riesgos en los productos desarrollados a efecto de realizar acciones de mitigación de aquellos (Clements *et al.*, 2008).

Un defecto es cualquier elemento en un artefacto (documentación o código) que provoque una falla en el sistema una vez que esté en operación, o bien, se refiere a que el sistema no cumpla con un criterio de calidad específico. La realización de sistemas es una actividad intelectual muy compleja, lo cual, de manera natural, la hace propensa a que se cometan o introduzcan defectos en ella.

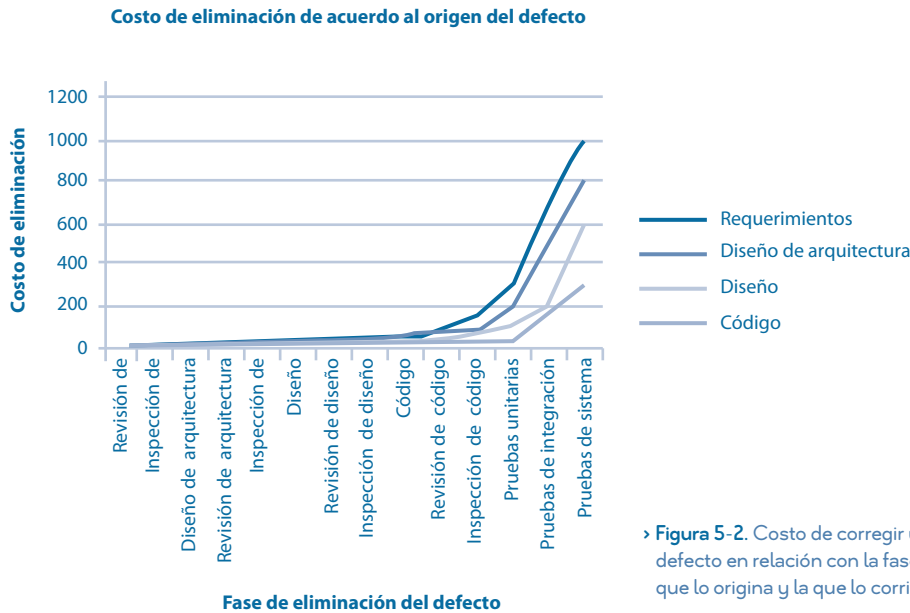
En el caso de las *arquitecturas de software*, los defectos se deben a elementos de un artefacto que provoquen una falla al sistema en operación, o bien, que este no cumpla con un *driver* arquitectónico específico.

5.3.1 Detección temprana de defectos en la arquitectura

El desarrollo de sistemas de *software* tiene una naturaleza incremental, de tal forma que si un producto intermedio contiene defectos, las siguientes actividades que lo usen como una entrada, se llevarán a cabo con una entrada incorrecta, generando productos intermedios erróneos y, por lo mismo, incrementando el número de defectos (se agregarán aumentos sobre bases erróneas). Tanto más tiempo pase entre la generación del defecto y su corrección, cuanto más serán los productos intermedios involucrados pues se da un efecto multiplicador.

Si el error es encontrado rápidamente, no hay productos adicionales por ajustar, solo el que lo contiene, por lo cual su corrección es menos compleja y, en consecuencia, más económica. La figura 5-2 incluye resultados de estudios de varios proyectos en donde se muestra el costo de eliminación de un defecto (en minutos), dependiendo de la fase en la que se le descubre y elimina.

Por su esencia, la arquitectura es la base para los diseños y la construcción del sistema. Encontrar defectos arquitectónicos cuando ya se tienen desarrollados varios componentes del sistema puede implicar costos serios de rehacer trabajo, así como retrasos en el proyecto.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Carreaga.

5.3.2 Satisfacción de los *drivers* arquitectónicos

Las evaluaciones deben siempre enfocarse en la satisfacción de los *drivers arquitectónicos* (requerimientos de los usuarios y de otros grupos involucrados con el sistema) pues son la base de la calidad del producto. En los productos terminados, los usuarios son las personas que utilizarán el sistema. En los productos intermedios no siempre son estas, sino quien va a emplear el producto intermedio; por ejemplo, un diseñador es usuario de los productos generados del análisis de requerimientos y, probablemente, de los del diseño de la arquitectura, los cuales espera que estén realizados en modo correcto además de, en esencia, libres de defectos.

5.3.3 Identificación y manejo de riesgos

Un riesgo es un evento posible en el futuro, que en caso de ocurrir tendrá un efecto sobre las metas del proyecto. Se caracteriza por la probabilidad de que ocurra y por el impacto que tendrá en las metas del proyecto. Una buena administración de proyectos incluye estar al pendiente de los riesgos muy probables con potenciales impactos significativos. Por su naturaleza, cualquier riesgo asociado a la arquitectura tiene casi siempre una consecuencia muy alta y negativa en las metas del proyecto.

Las evaluaciones permiten tener visibilidad sobre los riesgos más probables y costosos a los que pueda enfrentarse el proyecto. Esto facilita la toma de decisiones para evitar o disminuir el impacto de los riesgos, con lo cual se establecen estrategias proactivas de la administración del proyecto para que este pueda alcanzar sus objetivos.

En la tabla siguiente se muestran de manera general los riesgos más frecuentes y de mayor impacto en proyectos de desarrollo de *software*.

Categoría	Descripción
Requerimientos funcionales	Las funciones construidas en el producto no satisfacen lo que los usuarios necesitan.
Requerimientos no funcionales	El sistema es incapaz de satisfacer los atributos de calidad que espera el usuario.
Desconocimiento técnico	El personal no cuenta con el conocimiento necesario para plantear una solución técnica que satisfaga los requerimientos de los usuarios.

5.4 CARACTERÍSTICAS DE LOS MÉTODOS DE EVALUACIÓN DE ARQUITECTURAS

Existen varios métodos para evaluar los productos de *arquitectura de software*, de los cuales se distinguen:

1. *Por el momento en el que se realiza la evaluación.* Esto es, mientras se define la arquitectura (diseños parcialmente terminados), en el momento en que ya se terminó de definir (diseños terminados) o cuando ya se desarrolló el sistema basado en ella (sistema terminado).
2. *Por el personal que la realiza.* Personas que están desarrollando o desarrollaron la arquitectura, o bien, personal que no participó en su desarrollo.

En las siguientes subsecciones se describen con más detalle.

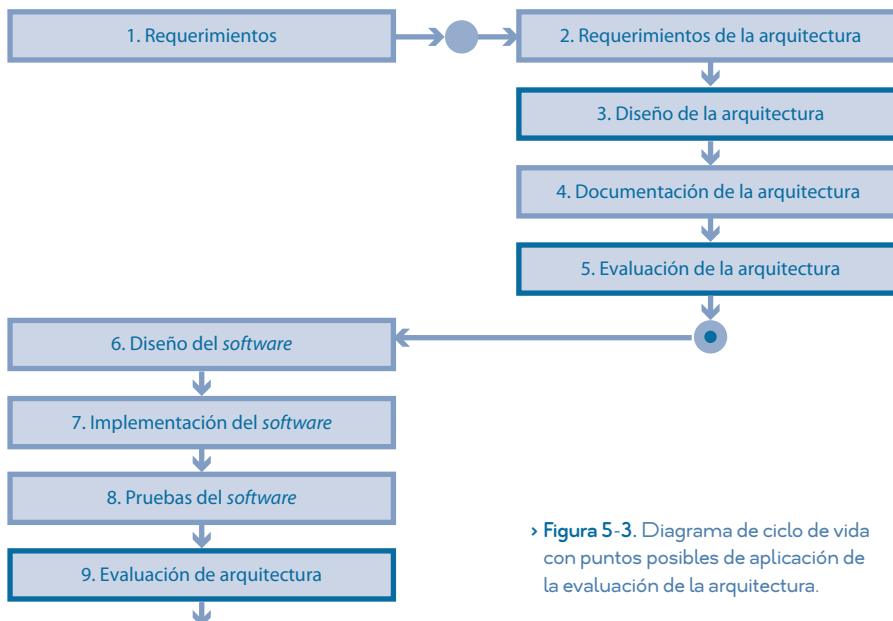
5.4.1 Producto que se evalúa: diseños o productos terminados

La evaluación de arquitectura se aplica tanto a diseños para detectar errores de manera temprana como a productos terminados para evaluar riesgos de colocar en producción el sistema terminado.

En el caso de la evaluación de diseños, por lo habitual se busca que estos sean correctos con base en la especificación disponible de los *drivers arquitectónicos*, y que los cubra de manera adecuada. Sin embargo, también se puede pretender que cumpla con las necesidades de los usuarios aun cuando estos no pueden ver un producto terminado.

En el caso de productos terminados, las evaluaciones buscan asegurar que satisfagan los *drivers* actuales, así como que el producto haya sido construido de manera correcta pues algunos defectos por incumplimientos de *drivers* pueden ser causados por una implementación errónea. Estas evaluaciones permitirán tomar acciones para mitigar riesgos referentes a poner sistemas en producción, o bien, respecto de sistemas que ya están en producción. La mitigación de riesgos puede incluir no salir a producción, generar proyectos de reemplazo, etcétera.

Los métodos de evaluación se incorporan en los procesos utilizados para el desarrollo del producto o de la arquitectura. En la figura 5-3 se resaltan las etapas en las cuales pueden realizarse evaluaciones de arquitectura.



› **Figura 5-3.** Diagrama de ciclo de vida con puntos posibles de aplicación de la evaluación de la arquitectura.



En el caso del ACDM visto en el capítulo 2, la evaluación se hace en la etapa 4: revisión de la arquitectura. En procesos como RUP, las evaluaciones están indicadas en cada paso hasta la finalización del producto.

5.4.2 Personal que lleva a cabo la evaluación

De acuerdo con sus principios, las evaluaciones de *arquitecturas de software* buscan asegurar que los productos satisfagan los *drivers arquitectónicos*. Por ello son indispensables evaluaciones en las que participen interesados en el sistema, por ejemplo, los usuarios finales, que son algunos de quienes definen los *drivers*. Sin embargo, si solo se llevan a cabo estas evaluaciones, los productos evaluados tienen por lo regular un número muy grande de defectos que ellas serán incapaces de detectar y eliminar, quedando latentes en el producto final, y por lo mismo es muy probable que este tenga problemas de calidad graves.

Lo anterior es una de las razones para que el personal encargado de desarrollar el sistema y, en particular, el de la arquitectura, realice evaluaciones para detectar de manera anticipada los errores en la arquitectura. Lo anterior hará que la evaluación del producto terminado hecha por aquellos interesados externos al equipo de desarrollo sea más efectiva, pues se asegura que el producto terminado ya cuenta con un número reducido de defectos.

El equipo de desarrollo del sistema cuenta además con el conocimiento de los elementos que componen la arquitectura y sus interrelaciones, de tal forma que puede evaluar elementos muy específicos y/o críticos de ella.

5.5 MÉTODOS DE EVALUACIÓN DE ARQUITECTURAS

Las evaluaciones de arquitecturas se realizan de distintas maneras: pueden ir desde revisiones simples hasta las que se hacen en talleres especializados de revisión, todo ello junto con la elaboración de experimentos y modelos que permitan evaluar si se cumplen o no determinadas características de la arquitectura.

Cada método de evaluación tiene ventajas y desventajas, siendo el contexto en el cual se usa lo que influye para obtener los mayores beneficios de los mismos. Durante el diseño de la arquitectura son más recomendables revisiones personales y ágiles, así como el desarrollo de experimentos o prototipos. Al final, convienen revisiones realizadas por personal externo al proyecto para detectar tanto situaciones en que el comportamiento del sistema no se dará como está previsto, es decir, *puntos de sensibilidad* de la arquitectura, como *relaciones de equilibrio* entre atributos de calidad que se contradicen entre sí y estos no han sido notados por el personal del proyecto.

Otras evaluaciones están inmersas en el proceso, como ocurre con el ACDM, el cual por su naturaleza iterativa plantea revisiones durante cada iteración de su proceso de diseño.

A continuación se describen los métodos más reconocidos para evaluar arquitecturas:

- Revisiones e inspecciones.
- Recorridos informales de diseño.
- ATAM (*Architecture Tradeoff Analysis Method*), o Método de evaluación de equilibrios de arquitectura.
- ACDM (*Architecture Centric Design Method*), o Método de diseño centrado en la arquitectura.
- ARID (*Active Reviews for Intermediate Designs*), o Revisiones activas para diseños intermedios.
- Desarrollo de prototipos o experimentos.

5.5.1 Revisiones e inspecciones

Las revisiones e inspecciones están entre las mejores prácticas de *ingeniería de software* para aumentar la productividad y la calidad de los equipos de desarrollo. Las primeras se asocian por lo regular a la evaluación hecha a un artefacto por la persona o personas que lo crearon. Las segundas tienen relación con la evaluación que se le hace a un artefacto por personas que no lo realizaron. En la figura 5-4 se muestran las entradas, salidas, roles de personas y herramientas que se manejan en las revisiones o inspecciones.



› **Figura 5-4.** Revisiones e inspecciones como caja negra (entradas y salidas).

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Caraga.

Las inspecciones en el contexto de desarrollo de *software* fueron definidas por primera vez por Fagan (Fagan, 1976). A partir de ese momento se ha visto como una de las mejores prácticas hasta el presente, existiendo gran número de referencias que las explican de manera detallada, como Wong (Wong, 2006).

Para que sean efectivas y logren sus metas de detección de defectos las revisiones e inspecciones, tienen que realizarse con un nivel de formalidad alto siguiendo un proceso definido y cumpliendo con criterios mínimos de calidad. Este tipo de evaluaciones no son frecuentes. En su lugar encontramos revisiones informales por parte de los propios diseñadores y arquitectos, las cuales aunque son muy útiles son menos efectivas que las formales.

Los procesos que las describen plantean criterios de entrada (artefacto a ser evaluado y listas de verificación), pasos a seguir (forma en que se realizarán la evaluación, la corrección de defectos y la verificación de que su eliminación satisfactoria) y criterios de finalización (artefacto evaluado y con todos los defectos eliminados).

Por lo regular, la forma de hacerlas es seguir una lista de verificación, tomar un elemento de esta y revisar el artefacto por completo, observando si cumple con ese elemento. En caso de que no cumpla, se marca como defecto y se corrige. Si lo cumple, el elemento se marca para indicar que el artefacto lo cumple. Se recomienda llevar a cabo primero una revisión personal del artefacto, y posteriormente una inspección de otros desarrolladores. Con la primera se eliminan defectos que podrían distraer fácilmente a los inspectores de encontrar errores más costosos en potencia.

Otra forma de efectuar una revisión o inspección es mediante el uso de los artefactos en lugar de solo hacerlas de manera estática. Implica que los artefactos que están en revisión se usen como está previsto; por ejemplo, si se toma una porción del diseño de la arquitectura, se utilizará para derivar el diseño de la solución e incluso para generar un módulo en código. Este tipo de revisión recibe el nombre de *revisión activa*, y es muy útil para encontrar defectos importantes que pueden tener gran impacto en las metas del proyecto de desarrollo.

Estudios de muchos proyectos que utilizan revisiones e inspecciones siguiendo procesos formales plantean que en el momento de la evaluación encuentran de 50 a 70% de los defectos (Jones, 2010). Estos resultados son mucho mejores que los de las pruebas tradicionales, las cuales encuentran entre 30 y 50% de los defectos. Revisiones e inspecciones más informales hallan también este número de fallas.

La cantidad de defectos que se detectan depende de varios factores:

- **Conocimiento del inspector acerca del artefacto a evaluar.** Es necesario que el inspector de un documento de diseño tenga conocimiento y experiencia en los aspectos técnicos que incorporan los artefactos de arquitectura; de otra manera, serán muy pocos o triviales los defectos que encuentren.
- **Velocidad a la que se hace la revisión o la inspección.** Se ha observado que mientras más rápido se revise o inspeccione un documento, menos defectos se encuentran. Un número genérico es revisar o inspeccionar a una velocidad máxima de cuatro páginas por hora.

Los artefactos a revisar o inspeccionar son: documentación de los *drivers*, escenarios de atributos de calidad, diseños de arquitectura, diagramas de arquitectura, documentos de descripción de la arquitectura, entre otros.

Los métodos de evaluación ATAM, ACDM en su etapa 4, y ARID son en realidad inspecciones ajustadas especialmente a *arquitecturas de software*.

5.5.2 Recorridos informales al diseño

El procedimiento más ampliamente usado es cuando el equipo de arquitectura hace una presentación del diseño de esta a otras audiencias. Se usa con frecuencia porque es muy fácil de realizar y no requiere de mucho esfuerzo; sin embargo, es demasiado limitada en su capacidad de encontrar elementos de riesgo de la arquitectura.

Es una presentación por lo regular de una a dos horas, la cual no tiene un proceso definido y puede usarse para mostrar el tipo de diseño que sea, incluido el arquitectónico. Los asistentes pueden o no ser personal técnico, además, no necesariamente se aplican escenarios de requerimientos funcionales o de atributos de calidad. Los asistentes hacen preguntas al equipo que llevó a cabo la arquitectura, y este da las respuestas correspondientes, registrando los problemas que pueden descubrirse en ella.

Aun con sus limitaciones, es un ejercicio ampliamente recomendado pues además de apoyar la evaluación del diseño de la arquitectura, también es un ejercicio de comunicación de esta.

Estos recorridos pueden mejorarse al incorporarles más elementos de formalidad, como los que pueden encontrarse en otros métodos, como el ATAM y la etapa 4 del ACDM, que se verán en las secciones siguientes.

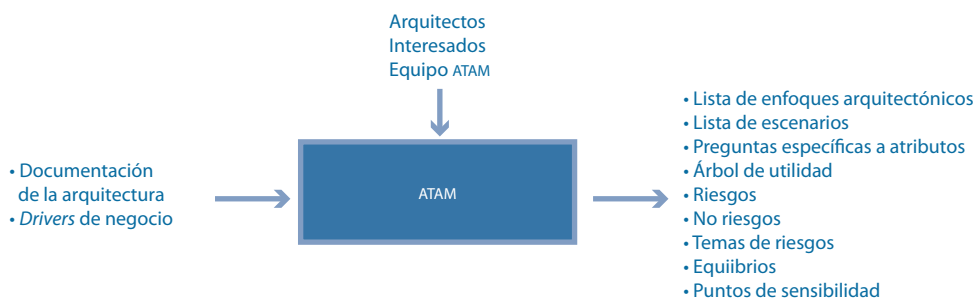
5.5.3 Método de análisis de equilibrios de la arquitectura (ATAM)

Uno de los procesos de evaluación para *arquitecturas de software* más conocidos es el Método de análisis de equilibrios de la arquitectura, o ATAM, (por sus siglas en inglés). Desarrollado por los investigadores del SEI, tiene como objetivo “evaluar las consecuencias de las decisiones arquitectónicas respecto de los requerimientos de *drivers* del sistema” (Kazman, Klein y Clements, 2000).

Su nombre se debe a que no solo se evalúa que una *arquitectura de software* cumpla con la calidad exigida, sino que analiza las interacciones de los distintos atributos de esta y cómo ellos se equilibran o restringen unos a otros. Puede realizarse más de una vez durante el diseño de la arquitectura, o bien, una vez concluido, y aplicarse además a productos de *software* terminados o en proceso. Es definido por Kazman, Klein y Clements (2000), y se explica con mayor amplitud en el libro de Clements, Kazman y Klein (*Evaluating Software Architectures*, 2002).

El ATAM toma como entradas la documentación de la arquitectura y cualquier otro artefacto que haya sido producido por el diseño de esta. El resultado de hacer una evaluación usando el método es una lista de riesgos en ella, puntos de sensibilidad o que requieren atención, y puntos de equilibrios entre atributos de calidad.

Los roles principales en una evaluación con el ATAM son el arquitecto de *software*, los interesados en el sistema con la arquitectura y el personal que apoya la realización de la evaluación con este método, conocido como equipo ATAM. La figura 5-5 ilustra las entradas, las salidas y otros aspectos del método.



► **Figura 5-5.** EL ATAM como caja negra (entradas y salidas).

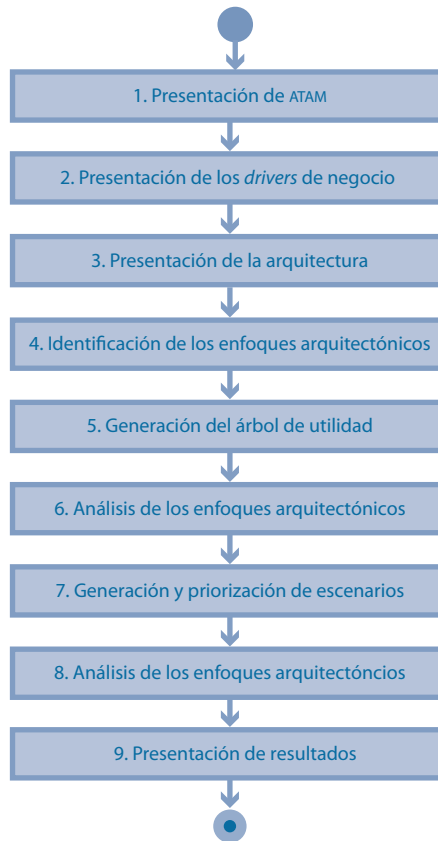
El método se propuso en un inicio para evaluar los diseños de la arquitectura, así como para detectar de manera temprana cualquier defecto o riesgo, y así fuera resuelto este antes de entrar al desarrollo del producto. Sin embargo, ha sido empleado además tanto en productos legados como en recién terminados para evaluar su arquitectura (cuando se cuenta con ella) y su nivel de cumplimiento de los *drivers*.

El ATAM es uno de los procedimientos más elaborados propuestos por el SEI, tanto que requiere de personal capacitado para poder guiar su práctica. Es común que las organizaciones que requieran una evaluación ATAM no dispongan de este personal, por ello deben apoyarse en personal externo. El SEI es una de las organizaciones que pueden ser contactadas para hacer evaluaciones de esta clase.

Tal situación es importante porque define la forma en que se llevan a cabo las evaluaciones ATAM. De manera cronológica, se da en cinco fases:

1. **Preparación de la evaluación.** Incluye todo el protocolo de contacto entre organizaciones, contratación y alcance de la evaluación. Dado que una evaluación con el ATAM requiere que participen distintos actores, es indispensable que sea planeada y se den las condiciones que permitan su adecuada realización. Por lo mismo, todas inician con una preparación, la cual consiste en determinar su duración más adecuada, asegurar que asistirán todos los interesados en el sistema, integrar los equipos, al igual que definir las fechas y cada uno de los aspectos de logística, como sala de juntas, suministros y herramientas (proyectors, pizarrones, cuadernos, etcétera).
Puede tardar varias semanas y se realiza principalmente por teléfono, videoconferencia y/o correo electrónico. Personal de contacto y los posibles líderes de cada equipo son quienes por lo habitual la llevan a cabo.
2. **Familiarización con la evaluación.** Es la primera parte de la evaluación, en donde por un lado el equipo ATAM se familiariza con el proyecto y la arquitectura, y por el otro, el equipo de la organización a ser evaluada hace lo propio con el método ATAM.
Durante esta familiarización se efectúa una evaluación preliminar y se prepara lo necesario (lista de interesados a participar, escenarios, árbol de utilidad, etc.) para efectuar la fase evaluativa. Se realiza por lo habitual en uno o dos días, y participa solo el equipo ATAM, así como el de arquitectura de la organización a ser evaluada.
3. **Ajustes previos a la evaluación.** La familiarización descubre elementos arquitectónicos que deben ser ajustados, tanto en forma y documentación, como en riesgos no considerados por el equipo de arquitectura. Lo anterior puede motivar a que este ajuste elementos antes de realizar la evaluación definitiva. Dependiendo de los cambios a realizar, la fase puede tomar desde unos minutos hasta varias semanas.
4. **Evaluación.** En esta fase se incorporan todos los interesados en el proyecto que estarán afectados por las decisiones de arquitectura. Ellos propondrán escenarios que deben ser cubiertos por esta y serán la base del análisis de la misma. Se identifican los riesgos más significativos, los equilibrios entre distintos atributos de calidad y los puntos de sensibilidad que ella pueda tener. La etapa toma por lo habitual un día o dos y participan los equipos ATAM e interno, así como los interesados en el proyecto.
5. **Elaboración del reporte final.** La última parte consiste en generar un reporte con todos los elementos identificados, con explicaciones y, en algunos casos, recomendaciones. También se recolecta y guarda la información a efecto de mejorar el proceso de ATAM para futuras evaluaciones. Se realiza por el equipo ATAM, y puede tardar varios días.

Las fases de familiarización y de evaluación se llevan a cabo con nueve pasos progresivos en los que se dan discusiones controladas para determinar si la arquitectura cumple o no con los requerimientos de atributos de calidad, y si cuando son incumplimientos deben ser considerados para efectos de corrección u otro tipo de decisiones respecto del producto. En la figura 5-6 se indican los pasos del ATAM.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

› Figura 5-6. Pasos del ATAM.

Los pasos pueden tener distintos participantes y objetivos, dependiendo de la fase del ATAM en la que se esté. A continuación se describen los pasos de una evaluación utilizando el ATAM para la fase de familiarización.

Fase de familiarización, paso 1: presentación del ATAM. El equipo ATAM presenta el método a los miembros del equipo de arquitectura: se describen sus objetivos, los pasos, las técnicas que serán utilizadas y los resultados esperados.

Fase de familiarización, paso 2: presentación de los *drivers* de negocio. El director del proyecto presenta a los miembros del equipo ATAM varios aspectos acerca del sistema en evaluación, lo cual incluye sus requerimientos relevantes, sus restricciones significativas, las metas de negocio que debe satisfacer cuando esté en producción, los involucrados principales y los elementos clave tomados en cuenta para el diseño de la arquitectura.

Esta presentación permite que el equipo ATAM tenga mejor entendimiento acerca del sistema, y los siguientes pasos de la evaluación sean realizados de manera adecuada. El equipo registra los *drivers* de la arquitectura.

Fase de familiarización, paso 3: presentación de la arquitectura. El equipo encargado de la *arquitectura de software* presenta al equipo ATAM la arquitectura generada durante el diseño, indicando sus aspectos principales, así como las restricciones tomadas en cuenta, por ejemplo, sistemas operativos o sistemas con los que ella interactuará, y cualquier otro aspecto que apoye las decisiones tomadas al diseñar. Esta presentación es de tipo técnico y el equipo ATAM registra las principales decisiones de arquitectura.

Fase de familiarización, paso 4: identificación de las decisiones de arquitectura. El equipo ATAM estructura y ordena toda la información obtenida en los pasos 2 y 3, para tener una base sólida para el análisis, resuelve dudas con el equipo de arquitectura, como las formas en que el sistema reaccionará ante escenarios de crecimiento, interconectividad, seguridad, etcétera.

Fase de familiarización, paso 5: generación del árbol de utilidad. Dado que son muy pocos integrantes del equipo ATAM y del de arquitectura, no es posible utilizar lluvias de ideas para obtener los escenarios con los cuales probar esta. Por ese motivo se usa una técnica la cual parte de la concepción de que la utilidad de un sistema se basa en los atributos de calidad.

Para ello se plantea un *árbol de utilidad*, el cual tiene precisamente a “la utilidad” como raíz, y las ramas son los atributos de calidad que contribuyen más a la utilidad del sistema. A su vez, cada atributo se ramifica en aspectos más específicos. El último nivel del árbol consiste en escenarios o ejemplos concretos de cómo se piensa que la arquitectura soporte esos aspectos.

Este árbol permite pasar de un nivel de abstracción alto de un atributo de calidad a escenarios concretos que la arquitectura debe realizar. A cada escenario se les asigna tanto un valor de prioridad para ser cumplido por esta, como uno de la complejidad técnica que representará efectuarlo.

El árbol es elaborado por el equipo ATAM con el apoyo del equipo de arquitectura. Crearlo es fundamental para la evaluación porque es el elemento con el que se establecen el alcance y las prioridades de los *drivers* que la arquitectura debe cumplir. Este paso es relativamente parecido a las actividades que se realizan durante el taller de atributos de calidad (QAW) descrito en el capítulo 2.

Fase de familiarización, paso 6: análisis de las decisiones arquitectónicas. Se analizan estas decisiones basándose en la información de los pasos 3, 4 y 5. El equipo de evaluación revisa los escenarios de mayor prioridad, el arquitecto resuelve dudas o hace aclaraciones, y se detectan de manera preliminar los riesgos, equilibrios y puntos de sensibilidad.

En este punto concluye la fase de familiarización. Podemos ver un ejemplo de ella en la sección 5 del caso de estudio del apéndice. Con los resultados preliminares del paso 6, el equipo encargado de la arquitectura puede desear hacer ajustes a esta.

La fase de evaluación iniciará una vez que se hayan hecho los ajustes y estén las condiciones para llevarla a cabo: instalaciones, logística y participantes. Los pasos se describen a continuación.

Fase de evaluación, paso 1: presentación del ATAM. El equipo ATAM presenta el método a los participantes. Se describen sus objetivos, los pasos, las técnicas que serán utilizadas y los resultados que se espera obtener de él.

Fase de evaluación, paso 2: presentación de los *drivers* de negocio. El director del proyecto presenta a los involucrados los aspectos más relevantes acerca del sistema en evaluación, lo cual incluye sus requerimientos importantes, restricciones relevantes, las metas de negocio que debe satisfacer cuando esté en producción, los interesados principales y los elementos más trascendentes tomados en cuenta para el diseño de la arquitectura. De esta manera, los participantes conocerán las razones y requerimientos clave del sistema a ser desarrollado.

Fase de evaluación, paso 3: presentación de la arquitectura. El equipo de arquitectura presenta a los participantes la *arquitectura de software* generada durante el diseño, indicando sus aspectos principales, así como las restricciones tomadas en cuenta: sistemas operativos, sistemas con los que interactuará y cualquier otro aspecto que apoye las decisiones tomadas al diseñar. Esta presentación es de tipo técnico, pero en un nivel que los participantes puedan entender. El equipo ATAM registra cualquier decisión arquitectónica nueva o cambio que pudo haber sucedido de lo que percibieron en este paso durante la fase de familiarización.



Fase de evaluación, paso 4: identificación de las decisiones arquitectónicas. El equipo ATAM expone a los participantes toda la información obtenida en la etapa de familiarización con los ajustes que pudieron salir en los pasos 2 y 3, a efecto de tener una base sólida para el análisis en los siguientes pasos.

Fase de evaluación, paso 5: presentación del árbol de utilidad. El equipo ATAM presenta a los asistentes el árbol de utilidad obtenido en la fase de familiarización. Se explica su estructuración, y se muestran los principales escenarios identificados, así como su importancia y la complejidad para su implementación.

Fase de evaluación, paso 6: análisis de las decisiones arquitectónicas. El equipo ATAM expone a los participantes las principales decisiones arquitectónicas identificadas en la fase de familiarización.

El equipo de evaluación revisa junto con los participantes los escenarios de mayor prioridad, y el arquitecto resuelve dudas o hace aclaraciones, se muestran los riesgos, equilibrios y puntos de sensibilidad identificados.

Fase de evaluación, paso 7: generación y priorización de los escenarios a ser considerados durante el análisis. El equipo ATAM se basa en el árbol de utilidad, y mediante una técnica de lluvia de ideas, guía en este punto a los participantes en la obtención de escenarios adicionales a los que el árbol de utilidad proporciona para que propongan otros, por ejemplo, de uso del sistema, así como de atributos de calidad en el uso futuro del mismo, como puede ser crecimiento, adaptaciones, etcétera. Después se hace una consolidación de los escenarios propuestos, eliminando los duplicados y determinando su importancia para el negocio por medio de una votación.

Cada escenario nuevo se coloca con su importancia en el árbol de utilidad, y el equipo de arquitectura determina la complejidad de implementación.

Fase de evaluación, paso 8: realización del análisis de los enfoques de la arquitectura basándose en los escenarios nuevos del árbol de utilidad. El equipo ATAM hace que el equipo de arquitectura describa cómo funcionará esta y resolverá los escenarios más significativos del árbol de utilidad.

Al hacer la descripción, el equipo de arquitectura permite a los asistentes conocer si existen puntos de sensibilidad, las relaciones de equilibrio entre atributos de calidad, o bien, riesgos en la ejecución de los escenarios. El equipo de evaluación documenta todos los hallazgos.

Fase de evaluación, paso 9: presentación del equipo ATAM de los resultados de la evaluación a los involucrados participantes. Este paso resume las entradas a la evaluación y los hallazgos de esta: puntos de sensibilidad, relaciones de equilibrio entre atributos de calidad y riesgos. También se describen las sugerencias para dar los pasos siguientes: ajustes a la arquitectura, o bien, elaboración de prototipos o experimentos adicionales.

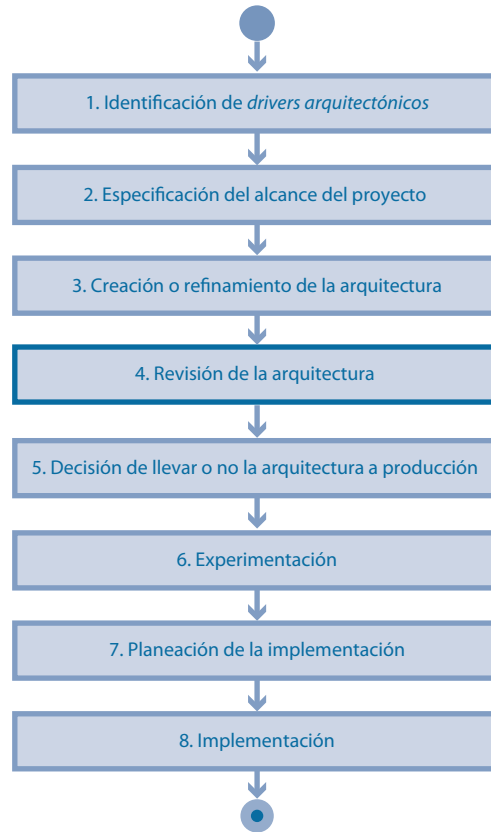
La última fase de una evaluación ATAM es la generación del reporte final. En ella, el equipo ATAM prepara y entrega este a la organización.

Después de una evaluación con el método, a los planes de proyectos debe incorporarse acciones que implementen las estrategias de mitigación de riesgos, y debe haber un seguimiento adecuado para asegurar que estos sean mitigados, o bien, que sean aceptados por todos los interesados del proyecto.

5.5.4 ACDM (etapa 4)

El método de diseño centrado en la arquitectura ACDM (Lattanze, 2008) fue presentando en la sección 2.3.2. En ella señalamos sus ocho etapas, las cuales volvimos a mostrar en la figura 5-7 señalando que la cuarta corresponde a la evaluación de la arquitectura.

El ACDM es un método integral, el cual se recomienda utilizarlo siguiendo estrictamente las etapas que propone, y eso hace diferente la evaluación respecto de los métodos evaluativos que pueden utilizarse junto con distintos procesos de desarrollo.



› Figura 5-7. Etapas del ACDM.

La etapa 4 del ACDM plantea que la evaluación puede ser realizada por el equipo de desarrollo o por uno externo y pone énfasis en que la arquitectura sea capaz de satisfacer los requerimientos y escenarios funcionales, sin dejar a un lado los de atributos de calidad.

Como es parte integral de un método que sigue todo el ciclo de vida de la arquitectura, las entradas para realizarlo son la especificación de *drivers* y el documento de diseño de la arquitectura. Su salida son problemas y riesgos detectados en ella para satisfacer los *drivers*.

La etapa 4 de revisión que hace el ACDM se lleva a cabo mediante un taller en el que participan distintos roles del equipo de desarrollo: equipo de arquitectura, equipo de requerimientos, soporte, calidad, procesos y operación.

El taller tiene las actividades siguientes:

1. **Introducción.** Los asistentes se presentan entre sí y se expone de manera general el ACDM, además de cómo se realizará la etapa 4.
2. **Revisión del contexto y restricciones de negocio.** Se presentan a los asistentes todas las consideraciones de negocio descubiertas en la etapa 1 y analizadas en la 2, como presupuesto, fechas comprometidas, recursos humanos, estructura organizacional del proyecto de negocio, comercialización y aspectos legales.
3. **Revisión de los requerimientos y las restricciones técnicas.** Se presenta a los participantes un breve recordatorio acerca de requerimientos funcionales y de los de atributos de calidad, al igual que de restricciones técnicas clave en el proyecto.

4. **Presentación de la arquitectura.** Se presenta a los asistentes la arquitectura que satisface los *drivers*, indicando cuáles son estos, así como estrategia de diseño, contexto del diseño, arquitectura obtenida, influencias externas o de sistemas legados, problemas principales y riesgos.
5. **Análisis funcional.** En este punto se toman los casos de uso y se revisa cómo la arquitectura debe manejarlos para que sean llevados a cabo. Cualquier situación que impida su realización se toma como un problema de la arquitectura.
6. **Análisis estructural.** Se toman escenarios de atributos de calidad y se revisa que la arquitectura pueda manejarlos. Cualquier situación que impida su realización se toma como un problema de la arquitectura.
7. **Resumen: revisión de los problemas encontrados.** En este paso se resumen los hallazgos encontrados en los pasos 5 y 6.

Después de la etapa 4 del ACDM, la etapa 5 plantea tomar la decisión de hacer otra iteración en las etapas 1 a 4, o bien, pasar a las siguientes etapas del método. La etapa 4 es muy efectiva pues integra de manera natural la evaluación de la arquitectura con la especificación de los requerimientos y el diseño arquitectónicos, así como con la toma de decisiones acerca de seguir mejorando el diseño arquitectónico o avanzar a las siguientes etapas del método.

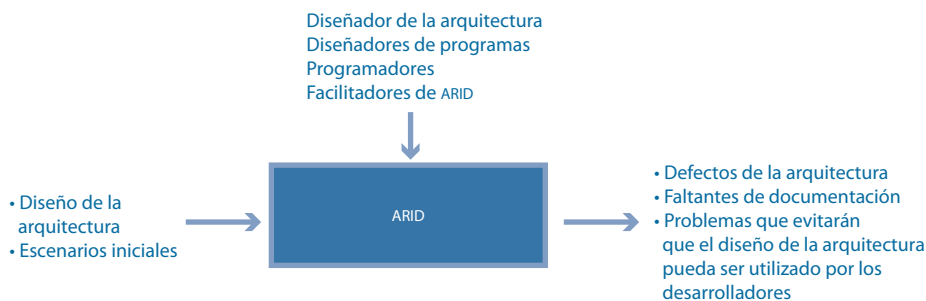
5.5.5 Revisiones activas para diseños intermedios (ARID)

Las revisiones llevan siempre el riesgo de que quienes se dedican a ellas hagan un trabajo deficiente, ya sea por tener un nivel técnico bajo o por una experiencia escasa respecto del producto que están evaluando. También puede ocurrir que para cuando se debe evaluar la arquitectura el proyecto ya está presionado, y las revisiones específicas, o no se hacen o son deficientes, o bien, son tardías.

Las revisiones activas plantean que los encargados de ellas deben ejecutar los productos a ser evaluados para detectar errores, excesos y carencias. Las realizadas a diseños evalúan productos de estos mediante la aplicación de situaciones que les permitirá a los evaluadores o revisores encontrar aspectos que tienen que ajustarse.

En determinadas situaciones es necesario conocer si los enfoques arquitectónicos, y en general el avance del diseño arquitectónico, son convenientes para diseño del sistema de *software*. En esos casos es necesario aplicar evaluaciones mientras se diseña la arquitectura para ver si desde el punto de vista de los diseñadores, la arquitectura es adecuada tanto en su concepción como en su documentación.

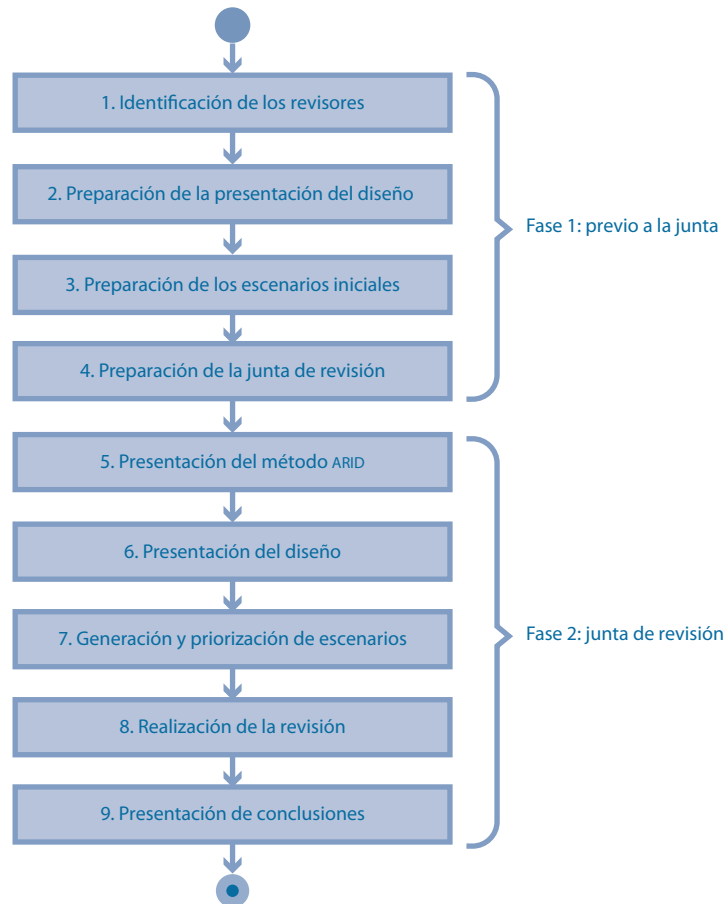
Las revisiones activas de diseños intermedios, ARID¹, por sus siglas en inglés, es un método de especialización de las revisiones activas aplicadas a diseños de *arquitecturas de software*, las cuales incluyen aspectos del ATAM. Se presenta en Clements (*Active Reviews for Intermediate Designs*, 2000) y se describe con mayor amplitud en Clements. *et al.* (2008) En la figura 5-8 se muestran sus entradas, salidas y participantes. Es notable que el ARID nombre a un *diseñador de la arquitectura*, sin embargo, puede considerarse idéntico al rol de *arquitecto* que se ha utilizado en los capítulos anteriores.



› Figura 5-8. El ARID como caja negra (entradas y salidas).

¹ *Active Reviews for Intermediate Designs*.

El método ARID plantea dos fases y nueve pasos; de estos, cuatro son actividades necesarias para tener la junta de revisión, y cinco se realizan durante la junta. En la figura 5-9 se muestran sus fases, y después se describen con mayor detalle.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

› Figura 5-9. Fases y pasos del ARID.

Fase 1: previo a la revisión

Paso 1: identificación de los revisores. Se selecciona a las personas idóneas para que participen en la revisión.

Paso 2: preparación de la presentación del diseño. El creador de la arquitectura se prepara para presentar el diseño de la misma a efecto de que personas con conocimiento técnico sean capaces de entender los principales componentes arquitectónicos y sus interacciones.

Paso 3: alistamiento de los escenarios iniciales. Se preparan escenarios para ilustrar los conceptos básicos de la arquitectura y facilitar así a los diseñadores de programas y/o programadores el entendimiento de estos. Los escenarios son ejemplos, y los revisores podrán utilizarlos para realizar la revisión o generar códigos de los que consideren más adecuados.

Paso 4: preparación de la junta de revisión. Se hace toda la logística de preparación de la junta: reproducción de materiales, invitaciones, reservaciones de salas de juntas, refrigerios, al igual que infraestructura, como proyectores, pizarrones, etcétera.



Fase 2: Junta de revisión

Paso 5: presentación del ARID. El facilitador presenta el método a los asistentes.

Paso 6: presentación del diseño. El creador de la arquitectura presenta a los asistentes el diseño de esta, teniendo como objetivo que entiendan sus características y las principales decisiones hechas en él. No hay discusión acerca de lo idóneo de las decisiones, solo se resuelven dudas acerca de su comprensión.

Paso 7: generación y priorización de escenarios. Al igual que en el ATAM, los asistentes sugieren escenarios y los priorizan. Esos escenarios serán aquellos sobre los cuales se realizará la revisión del diseño de la arquitectura.

Paso 8: realización de la revisión. Los asistentes toman los escenarios y el diseño de la arquitectura para intentar un diseño detallado de la solución que use la arquitectura y que resuelva el escenario. Ejercitar esto permitirá saber si la arquitectura es utilizable por los diseñadores de *software*, o conocer la identificación de aspectos complementarios, de clarificación, o de corrección de errores.

Paso 9: presentación de conclusiones. El último paso en el ARID es resumir todos los hallazgos encontrados durante la revisión para que sean resueltos por el diseñador de la arquitectura.

En general, tanto ATAM como ARID son muy parecidos; las diferencias se encuentran principalmente en el punto de vista que se tiene para la evaluación. En el primero se asume que la arquitectura es realizable como está y se evalúa el posible sistema final para ver que cumpla con los *drivers*. En el segundo no se asume que es realizable, sino que justo es lo que se va a evaluar, obteniendo hallazgos centrados en la descripción de la arquitectura y la coherencia técnica de esta.

En el ARID los revisores o evaluadores requieren tener el nivel técnico necesario para entender los aspectos detallados del diseño arquitectónico y plantear soluciones de programación que lo utilicen. En el caso del ATAM no es indispensable tal conocimiento.

5.5.6 Prototipos o experimentos

Los diseños de cualquier tipo plantean ideas que se implementarán posteriormente en programas que serán los constituyentes del sistema de *software*. El sistema construido mostrará de manera muy clara sus capacidades e incapacidades, pero su corrección será muy costosa.

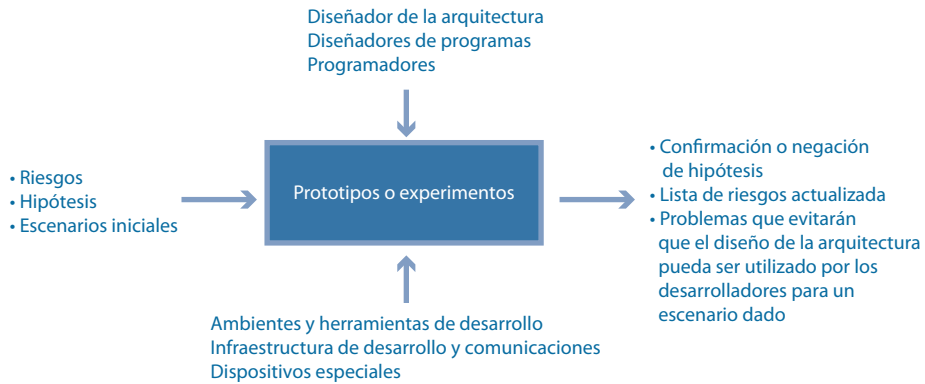
Los prototipos o experimentos buscan mostrar que las ideas de los diseños son o no realizables, permitiendo la adquisición de conocimiento que puede ser muy valioso sin tener que desarrollar el sistema por completo. También permiten obtener conocimiento sobre tecnología nueva o desconocida para el arquitecto, y observar así que su uso en la arquitectura será o no riesgoso.

Un prototipo es una solución muy reducida de desarrollo rápido que permite afirmar o negar las suposiciones en las cuales se basa la arquitectura. Un experimento es el planteamiento de una situación práctica con el que se confirma o niega una hipótesis en la cual se fundamenta la arquitectura.

En la figura 5-10 se muestran las entradas y salidas de prototipos o experimentos. La realización de ambos tiene como entradas los riesgos detectados durante el diseño, las hipótesis en las que están basadas las decisiones arquitectónicas y, finalmente, los escenarios relevantes que debe cubrir la arquitectura.

Los prototipos son propuestos por los arquitectos o diseñadores, y los llevan a cabo los desarrolladores. Utilizan herramientas e infraestructura de desarrollo y de comunicaciones, así como dispositivos especiales cuando lo requieren. Sus salidas son una conclusión que confirma o niega la hipótesis inicial, una lista de riesgos actualizada con base en las conclusiones y cualquier problema detectado en la arquitectura al momento de realizar el prototipo o experimento.

La elaboración de prototipos o experimentos es un mecanismo de verificación y/o validación de la arquitectura, pues permite conocer de manera práctica si esta es capaz de cumplir o no con determinados escenarios de atributos de calidad.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Carreaga.

› **Figura 5-10.** Desarrollo de prototipos como caja negra (entradas y salidas).

Los prototipos o experimentos proporcionan también conocimiento nuevo a los arquitectos acerca de las soluciones planteadas, de tal forma que pueden ajustarse para asegurar que la implementación de la arquitectura en un sistema se lleve a cabo con menores riesgos y de manera más eficiente: los prototipos, por ejemplo, sirven con frecuencia como punto de partida o elementos reutilizables para los equipos de desarrollo.

Los usos de los prototipos o experimentos son:

- Obtención de información acerca de riesgos arquitectónicos.
- Prospección tecnológica.
- Herramientas de aprendizaje.

Los resultados que se esperan son conocimiento nuevo, confirmación o rechazo de hipótesis y elementos básicos de reutilización.

Las limitaciones de los prototipos o experimentos son:

- La escala del prototipo o experimento nunca será la misma que la del sistema completo, lo cual puede generar imprecisiones en las conclusiones.
- Son actividades que deben formar parte del proyecto a efecto de que sean visibles para todos los involucrados.
- Son consumidores de recursos y tiempo del proyecto.

5.5.7 Comparación de métodos de evaluación

Ya revisados de manera general los métodos evaluativos, es conveniente tener un resumen de sus ventajas y desventajas y en qué contextos pudieran ser más útiles unos que otros. A continuación se muestra un cuadro comparativo de todos ellos.

	Revisiones e inspecciones	Recorridos informales al diseño	ATAM	Etapas 4 del ACDM	ARID	Prototipos o experimentos
Tipo	Mejor práctica	Mejor práctica	Método	Método	Método	Mejor práctica
Duración	Dependen del tamaño del artefacto a revisar. Típicamente cuatro páginas por hora (máximo)	Una a dos horas	Tres a cinco días	Medio día a dos días	Cuatro horas	Un día, máximo, por prototipo
Mecánica y enfoque	Revisión del artefacto apoyado en listas de revisión	Presentación del diseño de la arquitectura a una audiencia	Elección de <i>drivers</i> arquitectónicos, preparar escenarios y confrontarlos a la arquitectura con apoyo de facilitadores	Se aplican escenarios a la arquitectura con apoyo de facilitadores.	Se toma la arquitectura y se intenta utilizar para hacer acciones tanto de diseño como de programación y encontrar así deficiencias	Se toma una hipótesis y se diseña un experimento para asegurar su validez mediante el desarrollo de <i>software</i> que implemente el experimento
Provee un proceso que especifica las actividades, funciones y artefactos de entrada y salida	Sí (formales) No (informales)	No	Sí	Sí	Sí	No
Participantes	Quien elaboró el artefacto o colegas	Variados	Interesados, arquitectos y facilitadores	Interesados, arquitectos y facilitadores	Diseñadores y/o programadores	Diseñadores y/o programadores, así como arquitectos
Entradas	Cualquier artefacto a ser evaluado	Diseño de la arquitectura	Documentación del diseño de la arquitectura	• <i>Drivers</i> de la arquitectura • Documentación del diseño de la arquitectura	Documentación del diseño de la arquitectura	Porciones críticas del diseño de la arquitectura
Salidas	Lista de defectos encontrados y corregidos	Lista informal de observaciones	Lista de riesgos, puntos de sensibilidad y defectos encontrados	Lista de problemas de la arquitectura para toma de decisión	Lista de deficiencias del diseño de la arquitectura	Confirmación o negación de la hipótesis y puntos de sensibilidad
Criterios de Terminación	Todos los defectos detectados están corregidos	Se ha recorrido la arquitectura y no hay más preguntas	Identificados riesgos, no riesgos, puntos de sensibilidad y equilibrios	Se han aplicado todos los escenarios relevantes a la arquitectura	Tiempo límite y los defectos de la documentación de la arquitectura identificados	Prototipo funcionando o no en un tiempo límite
Contexto de utilidad	Cualquier documento o artefacto	Cualquier diseño	Diseños arquitectónicos de sistemas grandes y críticos que incluyan tecnología nueva	Diseños arquitectónicos de sistemas grandes y críticos que incluyan tecnología nueva	Factibilidad de diseños de arquitectura, y partes seleccionadas de ellos	Puntos específicos en los que se basan los riesgos más importantes de la arquitectura

EN RESUMEN

En este capítulo describimos las prácticas de las evaluaciones para asegurar que la arquitectura es correcta y completa, además de útil para su implementación por medio del diseño y la construcción de programas. La corrección se plantea de acuerdo con los *drivers arquitectónicos* identificados tanto en la etapa de requerimientos como en el momento de la evaluación.

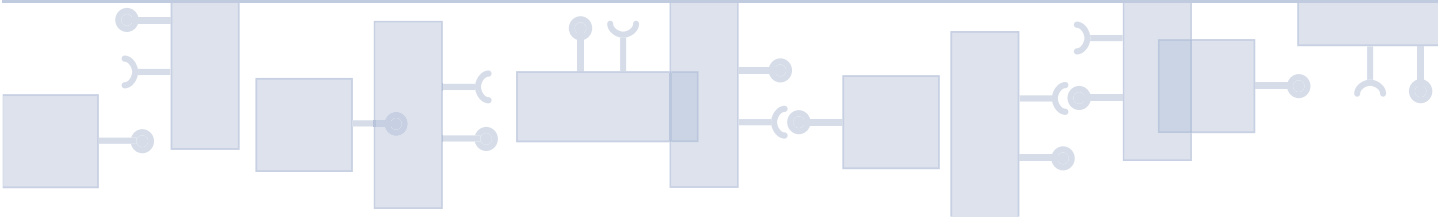
También mostramos las ventajas y beneficios de la detección temprana de defectos para evitar repetir trabajo en etapas posteriores del desarrollo. Examinamos distintos tipos de evaluaciones, algunas de carácter general como las revisiones e inspecciones, y otras configuradas especialmente para evaluar arquitecturas, por ejemplo el ATAM, la etapa 4 del ACDM y el ARID.

Mencionamos además que los prototipos y experimentos pueden considerarse como evaluaciones de conceptos o hipótesis sobre las cuales se fundamenta el diseño de la *arquitectura de software* de los sistemas.

Si un proyecto llega al punto de las evaluaciones de arquitectura, al realizarlas se podría asegurar que esta ya es correcta, completa y útil. Sin embargo, falta afianzar que la implantación del sistema se apegue a lo indicado en aquella. Este es el tema del que se ocupa el siguiente capítulo.

PREGUNTAS PARA ANÁLISIS

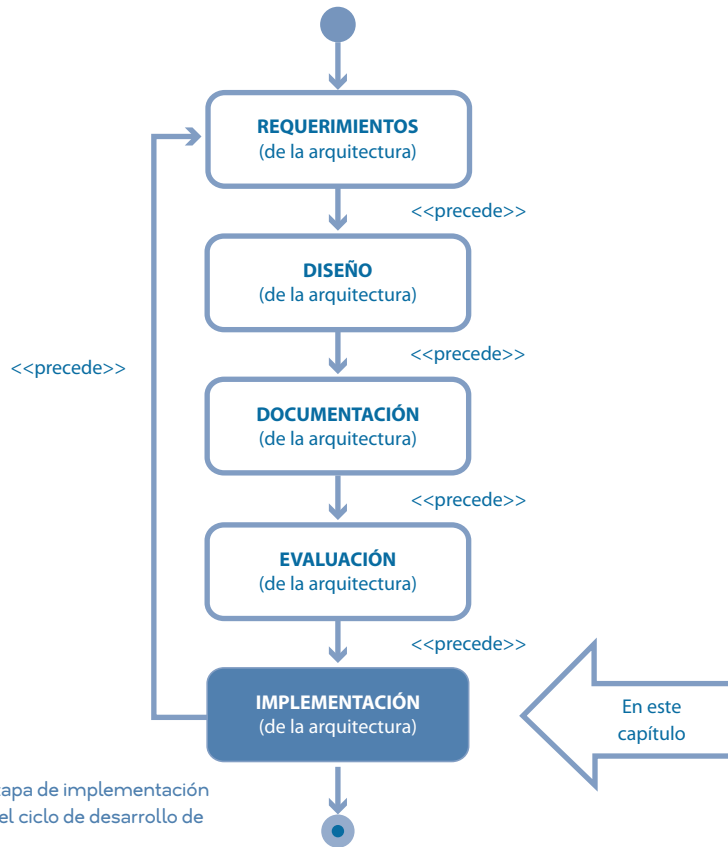
1. ¿Por qué es conveniente hacer una evaluación de arquitectura para un producto de *software* ya terminado?
2. ¿Cuál es la diferencia entre una verificación y una validación?
3. ¿Cuáles son más efectivas, las revisiones o las pruebas para la detección de defectos?
4. ¿Por qué es deseable una revisión personal antes de una inspección?
5. ¿Qué riesgos se asocian con las inspecciones?
6. Como una lista de revisión debe ser breve para que su aplicación sea efectiva, ¿cuáles criterios a revisar deberían integrarla en términos generales?
7. ¿Por qué el método ATAM se hace en dos etapas?
8. ¿Cuáles son las diferencias entre el método ATAM y el método ARID? Si en un proyecto debiera usted llevar a cabo tanto el uno como el otro, ¿cuál de los dos aplicaría primero y por qué?
9. ¿Cuáles son los riesgos que detecta principalmente el ARID?
10. ¿Qué riesgos elimina la creación de prototipos?



IMPLEMENTACIÓN: CONVERTIR EN REALIDAD LAS IDEAS ARQUITECTÓNICAS

Se piensa con frecuencia que el trabajo del equipo de arquitectura concluye cuando se tiene un diseño que ha sido evaluado y cubre satisfactoriamente con los requerimientos, pues se asume que el equipo de desarrollo implementará el diseño de la arquitectura tal y como fue especificado. Sin embargo, la falta de seguimiento durante la implementación del sistema puede hacer que el diseño de la arquitectura no quede plasmado adecuadamente.

El capítulo describe aspectos de la implementación influidos por la *arquitectura de software* y explica que sus discrepancias con esta son fuente de problemas durante el desarrollo y la operación del sistema. En la figura 6-1 se muestra el ciclo de desarrollo arquitectónico y se ubica la etapa de implementación dentro de este.



› **Figura 6-1.** La etapa de implementación del sistema en el ciclo de desarrollo de la arquitectura.

© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.

6.1 CONCEPTO DE IMPLEMENTACIÓN DE SOFTWARE

Las personas no relacionadas con los aspectos técnicos de los sistemas solo están interesadas en que estos les sean útiles y resuelvan sus necesidades de control y disponibilidad de información. La implementación de los sistemas genera el producto de *software* que será utilizado por los distintos usuarios.

La implementación se hace por lo habitual con base en arquitecturas preestablecidas, y las técnicas de diseño y construcción de sistemas se apegan a ellas. La implementación es la acción de transformar especificaciones en programas que serán funcionales para los usuarios.

La implementación comprende:

- Diseñar la estructura general del sistema basándose en la arquitectura. Lo anterior se refiere a identificar todos los módulos que permitirán soportar los requerimientos funcionales.
- Diseñar de manera detallada los módulos del sistema basándose en la arquitectura y los requerimientos funcionales.
- Desarrollar y/o adquirir los módulos especificados en el diseño.
 - › Programar código de cada uno de los módulos funcionales que no son adquiridos, de acuerdo a la especificación de sus interfaces.
 - › Incorporar cada elemento adquirido.
 - › Probar de manera individual cada uno de los módulos.

- Integrar los módulos desarrollados y/o adquiridos y probar que juntos funcionan como se espera.
- Probar los distintos niveles de integración hasta lograr la totalidad del sistema.
- Corregir cualquier error detectado en cada uno de los pasos.

Las actividades de implementación incorporan a la mayoría de los recursos humanos involucrados en el desarrollo de los sistemas.

6.2 LA ARQUITECTURA Y LA IMPLEMENTACIÓN DEL SISTEMA

La arquitectura es un conjunto de descripciones de la estructura de los elementos del sistema. Estas son abstractas y no se concretan hasta la implementación del sistema por medio de programas o elementos de *software* y *hardware* que interactúan entre sí.

Con frecuencia, el sistema resultante no cumple de manera adecuada con los *drivers*. Las razones pueden ser por una arquitectura errónea o porque los desarrolladores no siguen la arquitectura, o ambas.

Cuando la arquitectura no es la adecuada para los requerimientos del sistema, su implantación genera un producto que no satisface bien lo que necesitan los usuarios. Esto puede deberse a una deficiente identificación de requerimientos y *drivers* y/o a un diseño arquitectónico deficiente. Tales situaciones deberían identificarse por medio de una evaluación de la arquitectura, como se indicó en el capítulo 5.

El diseño de la estructura general en término de módulos y el diseño detallado de estos últimos deben considerar los elementos que señale la arquitectura. Cuando los desarrolladores no la siguen, crearán diseños de un sistema con una estructura no documentada y esta tendrá comportamientos no previstos desde el punto de vista de los *drivers*. Por lo mismo, la implementación de tales diseños constituirá un sistema que no se puede asegurar que satisfaga los *drivers*.

6.3 PRINCIPIOS DE LA IMPLEMENTACIÓN DE SOFTWARE

Para realizar la implementación del *software* deben seguirse algunos principios que aseguren el éxito de la implementación:

Principio 1. Generar diseños detallados de los módulos y otros elementos acordes con la arquitectura.

El diseño detallado de módulos y otros elementos debe apegarse a la especificación de las interfaces y ceñirse a lo indicado en la arquitectura del sistema, pues de lo contrario podrían impactar negativamente en la integración y en la satisfacción de los *drivers*.

Principio 2. Solo el equipo o persona responsable de la arquitectura puede realizar ajustes en esta.

En caso de encontrar errores, deficiencias o faltantes arquitectónicos, el equipo de implementación no debe intentar hacer las acciones de corrección o eliminación. Todas esas fallas deben notificarse siempre al equipo o persona responsable de la arquitectura para que sean resueltas y documentadas de manera adecuada.

Principio 3. Realizar la programación y/o adquisición de código existente acorde con los diseños detallados. Dado el principio 1, se acepta que los diseños detallados de los módulos y otros elementos cumplen con la arquitectura. Esto implica que toda la programación basada en estos diseños detallados cumplirá a su vez con los lineamientos que establece la arquitectura.

Principio 4. Ajustar los diseños detallados al encontrar errores o deficiencias. En caso de que la programación y/o la adquisición de código existente encuentre errores, deficiencias o faltantes en los diseños detallados de los módulos, el equipo de implementación debe hacer la resolución en estos antes de ajustar el código de los programas.



Al seguir los principios anteriores se eliminan los motivos de las inconsistencias entre implementación y arquitectura.

6.4 DESVIACIONES DE LA IMPLEMENTACIÓN RESPECTO DE LA ARQUITECTURA

Una *desviación* es la situación en que el diseño detallado de un módulo o su programación no son consistentes con las descripciones indicadas en la arquitectura. Las desviaciones provocan no asegurar que el sistema resultante satisfaga los *drivers*.

Se generan por el equipo de implementación y son de dos tipos:

- *La implementación corrigió errores, defectos u omisiones de la arquitectura, pero esta no fue corregida o ajustada.* Tales desviaciones tienen como consecuencia que las correcciones no fueron analizadas y diseñadas de acuerdo con criterios arquitectónicos, lo cual aumenta los riesgos de no satisfacer los *drivers*. De manera adicional, la documentación pierde valor pues no refleja la arquitectura real del sistema de *software*.

Estas desviaciones se dan con frecuencia por desarrolladores muy proactivos que no siguen los procesos de control de cambios, y también suceden por falta de comunicación o empatía entre el equipo de implementación y el de arquitectura.

- *La implementación ignoró elementos de la arquitectura.* Al igual que el tipo anterior, estas desviaciones tienen como consecuencia el incremento del riesgo de no satisfacer los *drivers* arquitectónicos una vez que la implementación esté concluida.

Estas desviaciones se dan por lo habitual por desarrolladores que desconocen la arquitectura o tienen poca disciplina para seguir las especificaciones arquitectónicas.

Las desviaciones son factores de riesgo del sistema al momento de colocarlo en producción. Cuando son detectadas, hay que tomar una decisión a efecto de resolverlas, o bien, no hacerlo. Este concepto forma parte de lo que se conoce como deuda técnica, ya que el proceso de toma de decisión sigue criterios financieros en relación con dejar que las deudas aumenten o sean pagadas.

Tanto más tiempo se mantengan las desviaciones, cuanto más aumentará la deuda técnica y, en consecuencia, el costo de retrabajar la arquitectura y/o la implementación de esta. El incremento en esa deuda se debe a los “intereses” que gana con el tiempo cuando no es resuelta.

Cuando se eliminan las desviaciones hay un costo asociado a rehacer el trabajo (retrabajo) de la arquitectura y/o implementación en el momento en que este se realice, incluidos los “intereses” ganados por la deuda técnica desde que ocurrió la desviación hasta que se corrige.

Los modelos de crecimiento de deuda mantienen una tendencia exponencial respecto del tiempo, de tal forma que en algún momento pueden ser tan grandes que con frecuencia la mejor decisión es no hacer la corrección y tomar los riesgos implícitos en las desviaciones.

6.5 PREVENCIÓN DE DESVIACIONES

Al hacer el análisis del manejo de la deuda técnica resulta claro que la mejor estrategia es la prevención de las desviaciones. Esta radica en evitar las causas que generan las desviaciones, de entre las cuales, las principales son:

- Falta de entrenamiento en los equipos de implementación.
- Carencia de seguimiento de los procesos de control de cambios.
- Comunicación deficiente entre el equipo de arquitectura y el de implementación.

6.5.1 Entrenamiento de diseñadores y programadores

El equipo de implementación está formado por personas que deben tener conocimientos técnicos necesarios para desarrollar los diseños detallados de los módulos y sus implementaciones de acuerdo con la arquitectura. Los conocimientos técnicos necesarios son establecidos por la arquitectura. Contar con los recursos humanos adecuados para la implementación depende del reclutamiento y la capacitación.

Respecto del reclutamiento, este debe considerar los conocimientos y habilidades necesarios para los distintos roles involucrados con la implementación. A efecto de asegurar que sea adecuado, se plantean prácticas como la revisión del currículo, entrevistas, exámenes técnicos y pruebas de habilidades verbales y matemáticas. Estas revelan deficiencias en los aspirantes, las cuales pueden o no ser resueltas con entrenamiento, y dan a conocer posibles riesgos relacionados con actitudes inadecuadas para el trabajo en equipo o las relaciones con los involucrados en los proyectos.

El entrenamiento es consecuencia de la detección de deficiencias en los recursos humanos, las cuales pueden ser técnicas, de conocimiento del problema a resolver, o bien, acerca de la arquitectura o los procesos que deben usarse durante la implementación. A efecto de tener entrenamientos efectivos se recomienda llevar una secuencia personalizada de estos en todos los miembros del equipo, y que cada entrenamiento tenga criterios de evaluación de los conocimientos y/o habilidades aprendidos.

Debe considerarse cualquier medio que pueda apoyar el aprendizaje: instrucción presencial o a distancia, *e-learning*, tutoriales, libros, presentaciones, seminarios, etcétera.

Es muy importante considerar que el equipo de implementación debe ser instruido en la *arquitectura de software*. Deben dedicarse tiempo y recursos para un adecuado entrenamiento acerca de ella. Una manera dinámica de hacerlo es que el arquitecto tenga un rol de mentor de la arquitectura para el equipo de implementación.

6.5.2 Desarrollo de prototipos o experimentos

Como se mencionó en el capítulo 5, los prototipos o experimentos son mecanismos para evaluar si un concepto arquitectónico funciona o no. En el ámbito de la implantación, aquellos que son realizados para la evaluación servirán como elementos de aprendizaje para el equipo que implementa, pues describen la forma en que determinados aspectos deben ser implementados de acuerdo con el punto de vista del equipo de arquitectura. Por ello es recomendable que todos los prototipos o experimentos estén disponibles para el equipo de implementación, con lo cual se previene que dicho equipo construya elementos del sistema conforme a su propia interpretación, evitando así las posibles desviaciones.

6.5.3 Otras acciones de prevención

Por lo habitual la principal acción preventiva es seguir los procesos para la obtención de una arquitectura y para hacer una implementación siguiéndola. En caso de no contar con un proceso, la primera acción es crearlo e implantarlo.

Los defectos de una arquitectura generada siguiendo de manera deficiente el proceso, o bien, sin proceso alguno, con frecuencia son fuentes de desviaciones durante la implementación. Un proceso adecuado y su realización disciplinada contribuyen a la prevención de desviaciones.

El método ARID descrito en el capítulo 5 apoya en buena medida la prevención pues permite identificar subespecificaciones en la documentación de la arquitectura, así como situaciones que serían desviaciones en potencia durante la implementación.

Otro elemento de prevención lo constituye el uso de *frameworks* arquitectónicos, que limitan a los equipos de implementación en el uso de elementos concretos de programación, de tal forma que no pueden desviarse de la arquitectura.

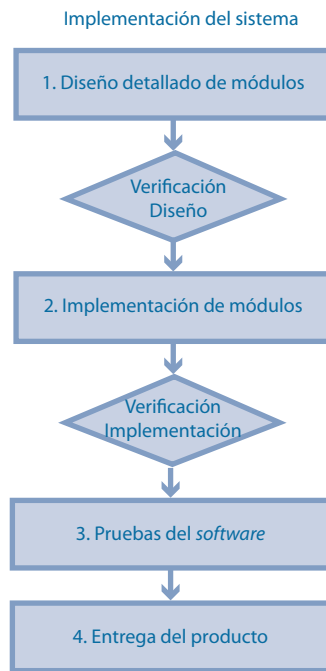
Los ambientes de desarrollo integrado (IDE, por sus siglas en inglés) incorporan los *frameworks* de tal forma que facilitan su uso y apego a la arquitectura. Existen también ambientes de diseño que tienen opciones de generación automática de código, los cuales producen los esqueletos de clases, interfaces y otros elementos indicados tanto en los diseños como en la arquitectura, y mantienen la alineación a estos.

Otra acción preventiva es la participación del equipo o persona responsable de la arquitectura durante la implementación del sistema de *software*. Este equipo o persona debe ser un mentor para aspectos de arquitectura: entrenando, resolviendo dudas, aclarando pormenores e incluso detectando errores en ella al momento de interactuar con el equipo de implementación.

6.6 IDENTIFICACIÓN DE DESVIACIONES: CONTROLES DE CALIDAD

Aun cuando se realicen actividades de prevención de desviaciones, es muy poco probable que estas no ocurran. Por ello es necesario tener procesos que apoyen a su detección temprana y evitar así costos altos por retrabajo. En la figura 6-2 se muestran las distintas etapas de implementación y en dónde pueden ejecutarse controles de calidad para encontrar las desviaciones.

En esta sección se describen brevemente algunas de las mejores prácticas para identificar desviaciones de la implantación respecto de la *arquitectura de software*.



» **Figura 6-2.** Ciclo de vida de diseño e implementación, y controles de calidad para la detección temprana de desviaciones.

6.6.1 Identificación de desviaciones durante el diseño y la programación

Como se indica en la sección 6.4, hay varias causas de las desviaciones de la implementación respecto de la arquitectura. En cualquier caso es muy conveniente detectarlas temprano durante la implementación para no agrandar los costos de un posible retrabajo. Las mejores prácticas son las verificaciones, las pruebas y las auditorías.

6.6.1.1 VERIFICACIONES DEL DISEÑO DETALLADO DE LOS MÓDULOS

Al finalizar un diseño detallado de los módulos y otros elementos debe comprobarse que no contenga defectos. La verificación puede hacerse por medio de revisiones personales o hechas por colegas, o bien, revisiones activas, las cuales describimos en el capítulo 5. Toda verificación debe considerar revisar que exista la alineación entre el diseño de los módulos y la arquitectura.

Es recomendable que personal del equipo de arquitectura participe en las verificaciones de los módulos y otros elementos críticos del sistema de *software* a efecto de ubicar de manera temprana las desviaciones.

6.6.1.2 VERIFICACIONES DE LA PROGRAMACIÓN

Al igual que con el diseño detallado, al finalizar la programación de uno o varios módulos es necesario hacer verificaciones para la detección de fallas y evitar que estas lleguen a las pruebas, donde será más costosa su localización y eliminación. Las verificaciones de programación pueden hacerse por medio de revisiones personales, revisiones por colegas o análisis estático automático de código.

En estas verificaciones se deben incluir revisiones de alineación del código con el diseño detallado del módulo, que a su vez debe estar alineado con la arquitectura.

6.6.2 Pruebas

Las pruebas son actividades relacionadas con la ejecución del *software*, que se basan en la aplicación de entradas específicas y obtención de resultados concretos seguidos de la comparación de estos últimos con los resultados esperados.

Lo anterior requiere de la existencia de dos momentos distintos para las pruebas. En el primero, estas se preparan mediante *casos de prueba*, los cuales describen objetivo, condiciones y procedimiento de la prueba, las entradas que deben proporcionarse y los resultados esperados. El segundo momento es cuando ya se dispone del *software*, o parte del mismo, y se le aplican los casos de prueba. La ejecución de un caso de estos plantea que los resultados reales del sistema se comparen con los esperados. Si son iguales, se considera que el sistema, o la parte del mismo, funciona bien para ese caso de prueba; si no, hay evidencia de la existencia de defectos, y deben ser corregidos.

Las pruebas pueden ser unitarias, en las que se verifican módulos aislados; de integración, en donde se examinan varios módulos al mismo tiempo, así como sus interfaces; de sistema, en las cuales se verifica el sistema completo; de desempeño, en las que se examina el atributo de calidad de desempeño, y de estrés, en donde se prueba el sistema en condiciones que rebasan los puntos de sensibilidad del mismo.

En cada una de ellas pueden estarse probando aspectos del sistema, los cuales tienen que ver con la arquitectura, o bien, detectándose situaciones de desviación entre ella y la implementación, o incumplimientos del sistema con los *drivers arquitectónicos*.

Las pruebas unitarias encuentran pocos defectos de arquitectura, sin embargo, pueden ayudar a ubicar desviaciones. Las otras pruebas apoyan la detección de defectos arquitectónicos, al igual que desviaciones.

Para asegurar que el sistema cumple con los *drivers*, las pruebas de sistema deben considerar casos basados en los escenarios de atributos de calidad generados como parte del QAW que se describe en el capítulo 2.



6.6.3 Auditorías

Las pruebas tienen una efectividad reducida para la detección de defectos y desviaciones, además de que exigen gran cantidad de esfuerzo y tiempo para su realización. En cambio, las auditorías son un mecanismo alternativo a las pruebas, más económico en esfuerzo y tiempo para localizar indicios de desviaciones o errores arquitectónicos. Se seleccionan de preferencia algunos módulos críticos y se analizan por expertos técnicos o personal del equipo de arquitectura buscando desviaciones. La auditoría solo busca indicios, si los encuentra, se hacen revisiones más detalladas a efecto de tener información más específica que sirva para la toma de decisiones.

La toma de decisiones se da respecto al manejo del riesgo del proyecto, el cual tiene que ver con decisiones de seguir o no seguir. En caso de no continuar, se debe optar entre si se va a retrabajar el módulo o se va a rehacer o, incluso, rehacer el sistema entero. Lo anterior proporciona visibilidad acerca del riesgo que puede existir en el sistema debido a las desviaciones.

El objetivo de las auditorías es tener de manera rápida elementos de información para la toma de decisiones, no tanto corregir las desviaciones detectadas. Las acciones pueden considerar hacer auditorías más detalladas, pruebas de estrés, desempeño o seguridad, rediseños, reprogramación de módulos, extensión de los periodos de prueba, etcétera.

6.7 RESOLUCIÓN DE LAS DESVIACIONES: SINCRONIZACIÓN DE LA ARQUITECTURA Y LA IMPLEMENTACIÓN

De acuerdo con lo señalado en el capítulo 5 y en la sección 6.4 sobre defectos no identificados oportunamente y la deuda técnica (errores identificados pero no corregidos), tanto más tiempo pase entre la generación de la desviación y su eliminación, cuanto mayor será el costo para resolverla. Por esta razón se busca eliminar cuanto antes las desviaciones o errores detectados. Esta resolución puede implicar correcciones a cada uno de los elementos: arquitectura, diseños detallados o programación.

6.7.1 Desviaciones en el diseño detallado

Resolver desviaciones en el diseño detallado de un módulo antes de que se realice la programación del mismo es más económico, pues al no tener artefactos generados, solo se tienen los costos de retrabajo del diseño.

Asumiendo que las desviaciones son que el diseño no respeta la arquitectura, las acciones de corrección deben ser rehacer los elementos del diseño que no se apegan a las especificaciones de la arquitectura. Estos elementos pueden incluir diagramas, descripciones de las responsabilidades e interfaces de componentes, casos de prueba, nomenclatura y/o pseudocódigo.

Las desviaciones deben quedar registradas de alguna manera para que el retrabajo necesario sea visible por medio de ajustes al plan de proyecto que incluya las actividades de corrección. Esto implicará un control de incidencias y un control de cambios.

6.7.2 Desviaciones en la programación

Cuando se identifican las desviaciones en la programación, estas son respecto del diseño detallado del módulo, el cual debe estar apegado a la arquitectura. Estas desviaciones no son evidentes en relación con la arquitectura, y se tratan por lo habitual contra el diseño. Cuando hay supervisión del equipo de diseño o del de arquitectura, son fácilmente manejables. Si esto no ocurre, posiblemente solo se detectarán hasta la fase de prueba y, por ende, puede darse en un momento en que el costo de retrabajo sea tan grande que la corrección implique gastos que salen del presupuesto original del proyecto, por lo que se tomarán otro tipo de decisiones.

La programación puede generar desviaciones por desconocimiento de aspectos de la arquitectura y por subespecificación o defectos en el diseño detallado de los módulos. Las correcciones necesarias se dan de acuerdo con la fuente de la desviación; si son de la programación respecto del diseño, entonces se corrige el código.

Cuando la desviación es producto de subespecificaciones y/o errores en el diseño, entonces hay que seguir el proceso de control de cambios para hacer los ajustes correspondientes en el diseño y tener un control de versiones adecuado. Esto puede provocar que algunos de los productos de la programación ya terminados requieran retrabajarse, pues fueron realizados con una versión del diseño que tenía errores y, por lo mismo, presentan también desviaciones.

6.7.3 Defectos y/o subespecificación en la arquitectura

Cuando el origen de las desviaciones se debe a errores en la arquitectura o a subespecificación en la misma, los diseños detallados de los módulos y su programación han sido elaborados sobre bases erróneas y se pierde en consecuencia el efecto unificador que la arquitectura tiene sobre ellos. Lo anterior da la posibilidad de generar varios diseños detallados de módulos que difieren entre sí en aspectos arquitectónicos.

El procedimiento a seguir debe ser el siguiente:

- El diseñador y/o desarrollador de los módulos encuentra un aspecto arquitectónico erróneo o subespecificado.
- Se notifica la situación al equipo de arquitectura.
- Se evalúa el impacto o deuda técnica asociada.
- En caso de que se decida eliminar la deuda técnica, se hacen las adecuaciones a la arquitectura correspondientes.
- En caso de que no se decida eliminar la deuda técnica, se documenta y reconoce su existencia.
- Si los cambios a la arquitectura son importantes, se vuelve a evaluar esta para asegurar que sigue cumpliendo con los *drivers*.
- Se elabora una versión arquitectónica nueva.
- Se comunican la versión reciente y los cambios a la arquitectura al diseñador y/o desarrollador de los módulos.
- El diseñador y/o desarrollador de los módulos retrabaja los diseños afectados por los cambios en la arquitectura y genera versiones nuevas de estos.
- En caso de ser dos entidades distintas, el diseñador de los módulos comunica a los desarrolladores tanto la versión nueva como los cambios en el diseño.
- El equipo de programación retrabaja los programas afectados por los cambios en los diseños de los módulos y genera versiones nuevas de estos.
- Se da por terminada la desviación.

Note que debe manejarse el control de cambios y de versiones en todos los artefactos implicados con la desviación.

6.7.4 Cuándo conviene no resolver las desviaciones

En caso de que la deuda técnica sea muy grande, y por los tiempos del proyecto y los recursos disponibles sea imposible resolver las desviaciones, la decisión puede ser aceptar las desviaciones con sus riesgos y costos asociados. Decisiones de ese tipo se realizan por el equipo de dirección del proyecto y los usuarios/clientes, pues pueden estar involucrados aspectos de manejo de contratos y otras obligaciones.



Tomar la decisión implicará seguir una serie de pasos:

- Determinar las acciones necesarias para evitar que la desviación se multiplique en otros diseños de módulos.
- Determinar los riesgos adquiridos por mantener la desviación.
- Determinar acciones requeridas para manejar los riesgos una vez que el sistema sea puesto en producción, por ejemplo, horarios especiales de operación de ciertos módulos y/o determinados tipos de usuarios, adquisición de equipos o espacio de almacenamiento, aumento del personal de soporte, etcétera.
- Incorporar las acciones en los planes de trabajo para que sean realizadas al momento en que el sistema sea puesto en producción.

Acciones adicionales pueden ser: crear nuevas fases del proyecto para generar versiones que resuelvan parcial o totalmente las desviaciones en el mediano plazo, renegociaciones de contratos, entre otras.

EN RESUMEN

En este capítulo presentamos las consideraciones que deben tenerse una vez que el desarrollo del sistema procede a partir de una arquitectura. En primer lugar establecimos en qué consiste la implementación del *software* y la relación que tiene con la arquitectura.

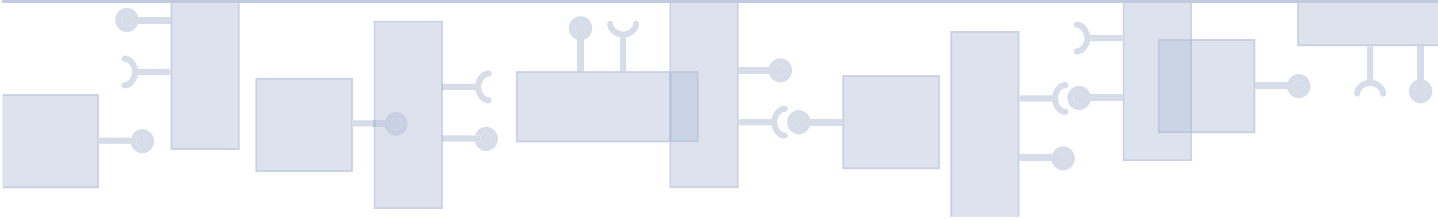
A continuación definimos los conceptos de desviación y deuda técnica que influyen en la relación entre la implementación del *software* y la arquitectura de este.

Por el efecto adverso que tienen, las desviaciones deben evitarse, y por ello indicamos distintas estrategias que previenen o minimizan su aparición. Sin embargo, es posible que ocurran, por ello describimos las distintas prácticas para identificarlas durante la implementación del *software*.

De manera posterior planteamos las acciones necesarias tanto para resolver las desviaciones como para no hacerlo, dependiendo de su naturaleza.

PREGUNTAS PARA ANÁLISIS

1. Suponiendo que se tiene una arquitectura correcta y especificada de modo adecuado, ¿cuáles son los riesgos arquitectónicos más importantes durante la implementación?
2. ¿Qué tipo de actividades debe realizar el equipo de arquitectura durante la implementación?
3. Suponga una situación en la que el equipo de *arquitectura de software* concluye y entrega esta al equipo de implementación y luego se desentiende del proyecto. ¿Cuáles son los riesgos asociados a este comportamiento?
4. Considere una situación hipotética en la que un miembro del equipo de implementación encuentra un defecto arquitectónico, lo corrige a efectos de lo que él está realizando y continúa con su trabajo. ¿Cuáles son los riesgos relacionados con este comportamiento?
5. Suponga que bajo el criterio de aprender sobre la marcha, en un proyecto se contrata a un equipo de implementación que no tiene experiencia con los elementos descritos en la *arquitectura de software*. Se le indica implementar de inmediato el sistema sin apoyo alguno del equipo encargado de la *arquitectura de software*. ¿Cuáles son los riesgos asociados a este comportamiento?
6. ¿En qué situaciones no es posible el pago de la deuda técnica?
7. Suponga que una auditoría de *arquitectura de software* hecha a 20 módulos detecta que dos tienen desviaciones significativas respecto de esta. ¿Qué toma de decisiones sería la más adecuada?
8. Considere una situación hipotética en la que se hallan desviaciones significativas en relación con la arquitectura durante el diseño detallado. ¿Qué toma de decisiones sería la más adecuada?
9. Suponga que se detectan desviaciones significativas respecto de la arquitectura durante las pruebas de aceptación. ¿Qué toma de decisiones sería la más adecuada?



ARQUITECTURA Y MÉTODOS ÁGILES: EL CASO DE *SCRUM*

En el contexto de la *ingeniería de software*, los métodos ágiles son un conjunto de métodos ligeros muy utilizados actualmente para soportar el desarrollo de sistemas. Su naturaleza ligera hace que sean poco prescriptivos, lo cual genera en muchos casos que los aspectos relacionados con el desarrollo de la arquitectura sean poco claros para quienes los practican. En este capítulo nos enfocamos en el *Scrum*, un método ágil muy popular ahora. Discutiremos en particular cómo actividades del ciclo de desarrollo de la arquitectura pueden realizarse en este contexto.

7.1 MÉTODOS ÁGILES

En la *ingeniería de software*, los métodos ágiles son métodos de desarrollo de sistemas con un enfoque iterativo e incremental. Sin embargo, en contraste con los métodos tradicionales que tienen este mismo enfoque, se caracterizan por equipos de personas multifuncionales (los integrantes tienen las habilidades necesarias), auto-organizados (quienes los integran requieren poca dirección), así como por iteraciones de desarrollo cortas en tiempo en las que siempre se produce una parte operable del producto final.

En el año 2001, antes de que se acuñara formalmente el término métodos ágiles, un grupo compuesto por críticos de los enfoques de desarrollo de *software* basados en procesos se reunió para definir las características que distinguen a los métodos ágiles de otros métodos. De esta reunión resultó el *Manifiesto ágil*, que contiene el conjunto de valores y principios que deben atender los métodos ágiles (Beck *et al.*, 2001). Los valores en el manifiesto son:

1. Individuos e interacciones por encima de procesos y herramientas.
2. *Software* funcionando por encima de documentación extensiva.
3. Colaboración con el cliente por encima de la negociación contractual.
4. Respuesta ante el cambio por encima de seguir un plan.

De estos cuatro valores se derivan los siguientes doce principios:

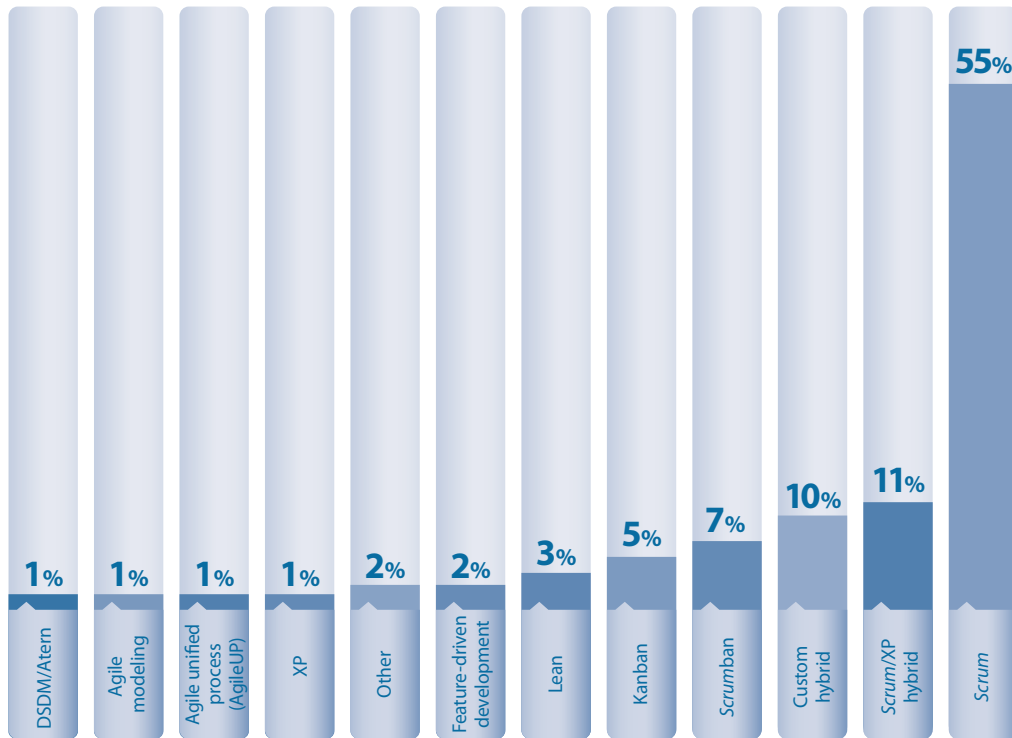
1. Satisfacer al cliente por medio de la entrega temprana y continua de *software* funcional.
2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo.
3. Entregar con frecuencia *software* funcional, prefiriendo periodos de semanas y no de meses.
4. Trabajo conjunto y cotidiano entre clientes y desarrolladores.
5. Construcción de proyectos en un ambiente con individuos motivados, dándoles la confianza y el respaldo que necesitan.
6. Comunicación cara a cara como forma eficiente y efectiva de comunicación.
7. El *software* funcional como principal medida de avance.
8. Procesos ágiles como medio para promover el desarrollo sostenido.
9. Atención continua a la excelencia técnica y al buen diseño.
10. Simplicidad como el arte de maximizar la cantidad de trabajo que no se hace.
11. Equipos de desarrollo auto-organizados.
12. Adaptación regular a circunstancias cambiantes.

Además de estos valores y principios en el manifiesto, los métodos ágiles se apoyan en diversas prácticas utilizadas durante el desarrollo de sistemas. Algunas relevantes incluyen el modelado dirigido por historias, programación en pares, desarrollo guiado por pruebas (Beck, 2002) o integración continua (Duvall, Matyas y Glover, 2007).

Como se nota, la filosofía de los métodos ágiles tiene mucha flexibilidad y dinamismo y difiere de aquella en la que el desarrollo de un proyecto realiza estimaciones, planes y diseños que, en general, son estables y generados desde las primeras etapas.

En la actualidad se observa un incremento en la adopción de métodos ágiles, además de que hay varios estudios en los cuales se reporta que su uso ha contribuido a reducir la complejidad y el riesgo presentes en varios métodos tradicionales de desarrollo (Cohn, 2009). De manera similar, se reporta que los métodos ágiles ayudan a lograr un mejor tratamiento de requerimientos cambiantes, aumentar la productividad de los equipos y mejorar la satisfacción de los clientes.

Existen varios métodos ágiles. La figura 7.1 muestra una gráfica con los más populares de acuerdo con un reciente estudio anual publicado por VersionOne (VersionOne, 2013). En el resto del capítulo nos enfocaremos en el que se reporta como el más popular: *Scrum*. Primero describiremos el método, y posteriormente discutiremos cómo actividades del ciclo de desarrollo de la arquitectura pueden llevarse a cabo en este contexto.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Cariega.

» **Figura 7-1.** Los métodos ágiles más utilizados de acuerdo con el estudio anual publicado en 2014 por VersionOne.

7.2 SCRUM

Scrum es un método ágil que puede aplicarse a casi cualquier proyecto para dar soporte a la administración de este. Es muy conocido en la actualidad por su uso en proyectos de desarrollo de *software*. Es en este contexto en el que se realiza nuestra descripción del método en este capítulo.

Como todo método ágil, *Scrum* permite de forma iterativa e incremental el desarrollo de *software*. Atendiendo al *Manifiesto ágil*, en *Scrum* un equipo multifuncional y auto-organizado crea de manera gradual un producto en varias iteraciones cortas. Cada iteración permite inspeccionar el rendimiento del equipo, así como el producto resultante, para luego, si es necesario, llevar a cabo oportunamente las adaptaciones requeridas.

Scrum define un conjunto pequeño de roles y un proceso que describimos en las siguientes secciones.

7.2.1 Los roles

Scrum define tres roles principales:

Propietario del producto (*product owner*): responsable de maximizar tanto el valor del producto que se está construyendo como el trabajo del equipo de desarrollo. Entre sus principales funciones están definir,



(re)priorizar y comunicar los requerimientos del producto, así como revisar y aprobar el trabajo y los resultados correspondientes en cada iteración. El propietario debe interactuar activa y regularmente con el equipo.

Equipo de desarrollo (*team*): grupo multifuncional y auto-organizado de personas encargadas de la construcción del producto. Entre sus principales responsabilidades está decidir el número de requerimientos a desarrollar en una iteración, así como la estrategia para llevarlos a cabo. El equipo se compone por lo habitual de entre cinco y nueve personas y no existen roles más específicos para los integrantes (por ejemplo el de programador o de diseñador).

Maestro *Scrum* (*Scrum master*): responsable de asegurar que el marco de trabajo de *Scrum* sea entendido y adoptado por todos los involucrados. Puede verse de esta forma como un facilitador para el propietario del producto y el equipo de desarrollo. En contraste con roles como el de líder de proyecto, que podrían parecer análogos, el maestro *Scrum* no indica al equipo qué se debe hacer en cada iteración. El maestro *Scrum*, sirve de ayuda al propietario y al equipo en el sentido que les ayuda a eliminar dificultades durante el proceso de desarrollo y promover el buen uso de prácticas ágiles de trabajo.

7.2.2 El proceso

El proceso de *Scrum* comprende un conjunto de iteraciones, las cuales tienen una duración fija, no pueden ser extendidas y se realizan una tras otra, sin interrupciones, hasta que el proyecto se considera terminado.

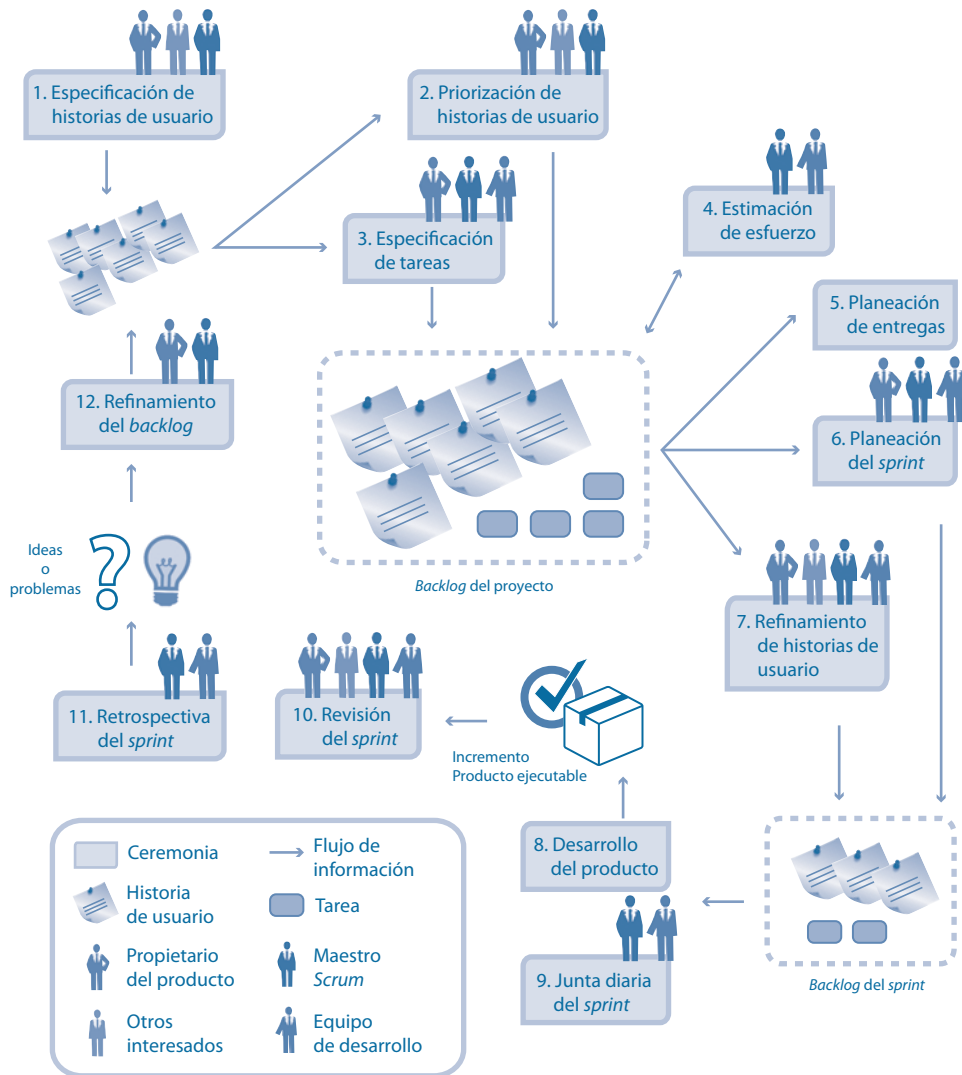
En *Scrum*, una iteración recibe el nombre de *sprint*, y su duración oscila por lo habitual entre una y cuatro semanas. En cada uno de ellos se lleva a cabo un conjunto de ceremonias organizadas en un patrón de actividades del tipo planeación-desarrollo-demostración-retrospectiva. Sin embargo, hay otras ceremonias que requieren realizarse antes del primer *sprint*. La figura 7.2, que describimos a continuación, muestra todas ellas, así como sus participantes, su secuencia y los principales artefactos producidos.

Como lo explicamos antes, *Scrum* define únicamente los roles de propietario del producto, equipo de desarrollo y maestro *Scrum*. Sin embargo, como se aprecia en la figura, en algunas ceremonias del proceso es productivo contar con la participación de otros interesados en el desarrollo del sistema (por ejemplo los usuarios finales). El maestro *Scrum* está presente en todas las ceremonias.

Al igual que cualquier proyecto de desarrollo de *software*, el proceso de *Scrum* inicia con la identificación de los requerimientos del sistema. El maestro *Scrum* se reúne con el propietario para especificarlos y priorizarlos (ceremonias 1, especificación de historias de usuario, y 2, priorización de historias de usuario), y en estas ceremonias podría requerirse la participación de otros interesados. En *Scrum* los requerimientos pueden ser determinados como historias de usuario, las cuales, como explicamos en el capítulo 2, son especificaciones cortas expresadas en el lenguaje del usuario final. La priorización de tales historias se orienta al propietario del producto debido a que, en esencia, se considera la relevancia de ellas en la satisfacción de los objetivos de negocio del sistema. De esta forma la priorización podría denotar, por ejemplo, el orden en el cual el propietario del producto desea que le sea entregada la funcionalidad.

El equipo de desarrollo toma en cuenta las historias de usuario a efecto de especificar una lista de tareas relacionadas (ceremonia 3, especificación de tareas). Esto se debe a que, para que a una historia se le considere como terminada, no solo hay que codificarla, sino que tiene tareas relacionadas como analizar o preparar la infraestructura necesaria para su implementación y pruebas, documentarla en los manuales correspondientes, generar su instalador en los equipos del cliente, etcétera. De ser necesario, el propietario del producto podría participar en la especificación de tareas. Las historias de usuario y tareas resultantes, como se ilustra en la figura 7-2, conforman lo que se conoce como *backlog* del proyecto.

En *Scrum*, el esfuerzo para completar una historia de usuario, y sus tareas asociadas, se especifica mediante *puntos de la historia* que son medidas relativas de complejidad que el equipo de desarrollo asigna en una ce-



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Careaga.
Ilustraciones: © Gamegix, PureSolution, Spiral media / Shutterstock.

»Figura 7-2. Principales ceremonias y artefactos del proceso de Scrum.

ceremonia posterior del Scrum (la 4, estimación de esfuerzo). Para el caso de las historias de usuario, la medida de esfuerzo debería establecerse considerando tareas de diseño, codificación y pruebas. Una vez estimado el trabajo requerido para completar los elementos del *backlog*, este podría ser revisado nuevamente por el propietario pues los resultados de la estimación podrían cambiar su visión acerca de las prioridades que él asignó antes, en la ceremonia 2.

Considerando la información en el *backlog* del proyecto en este punto, el propietario y el equipo de desarrollo acuerdan aspectos generales relacionados con las entregas (ceremonia 5, planeación de entregas). Esto es, acuerdan sobre cómo es que el Propietario del Producto irá obteniendo incrementalmente del equipo la funcionalidad esperada. Considerando el número de elementos en el *backlog*, así como las prioridades que tienen y el esfuerzo *requerido* para completarlos, también se establece la cantidad de *sprints* del proyecto y su duración, además de aspectos relacionados con el costo del proyecto.



Podemos decir que en este punto inicia un *sprint* de *Scrum*, y lo que se describe a continuación es un conjunto de ceremonias organizadas en el patrón de actividades planeación-desarrollo-demostración-retrospectiva que mencionamos antes.

La planeación consiste en que el propietario del producto y el equipo de desarrollo acuerdan qué elementos del *backlog* del proyecto se van a desarrollar en el primer *sprint* (ceremonia 6, planeación de *sprint*). De manera ideal, las historias de usuario y tareas relacionadas se seleccionan considerando la prioridad definida por el propietario y la complejidad de la implementación (denotado por la medida de esfuerzo determinada en la ceremonia 4). En *Scrum* es importante que el conjunto de elementos seleccionados pueda ser implementado en un *sprint*, lo cual se determina por lo habitual considerado la *velocidad* del equipo de desarrollo. La velocidad es una medida que establece el número de puntos de la historia completados por un equipo en un *sprint*. Si bien, la medida fluctúa en los *sprints* iniciales, tiende a estabilizarse después del tercero. Si la velocidad del equipo es menor al número de puntos de historia a completar en el primer *sprint*, se debe realizar una descomposición de las historias de usuario o tareas considerando una menor granularidad. Las historias y tareas a desarrollar en el *sprint* conforman lo que se conoce como *backlog* del *sprint*.

Antes de iniciar las actividades de desarrollo es necesario definir los criterios de aceptación de las historias de usuario. De esta forma, el propietario del producto y el equipo de desarrollo trabajan en la definición de las pruebas que deben realizarse a los elementos del *backlog* que las requieran (ceremonia 7, refinamiento de historias de usuario).

El desarrollo se refiere a la implementación del producto. Esto lo efectúa el equipo durante un periodo de varios días según la duración del *sprint* (ceremonia 8, desarrollo del producto). Como ya lo mencionamos, para el caso de las historias de usuario la implementación del producto incluye tareas de diseño, codificación y pruebas. Durante el desarrollo, el equipo se reúne diariamente para inspeccionar el progreso y, en su caso, realizar los ajustes necesarios para completar el trabajo restante en tiempo y forma (ceremonia 9, junta diaria del *sprint*). Si el equipo completara el trabajo antes de terminar el *sprint*, podría llevar a cabo más actividades, por ejemplo, completar el siguiente elemento más importante en el *backlog* del producto.

Una vez concluido el periodo de desarrollo del producto, inicia la demostración. El equipo se reúne con el propietario y, si es conveniente, otros involucrados relevantes, para revisar y demostrar lo que se ha construido (ceremonia 10, revisión del *sprint*). Como lo hemos mencionado, dependiendo del número del *sprint*, la revisión puede ser de un incremento funcional del producto, o bien, del producto final. Las historias de usuario aprobadas y las tareas completadas satisfactoriamente son removidas del *backlog* del producto.

Después de la demostración sigue la retrospectiva. El equipo discute sus impresiones acerca del trabajo realizado durante el *sprint* (ceremonia 11, retrospectiva del *sprint*). Como se ilustra en la figura 7-2, en esta reunión se genera un conjunto de ideas de mejora o problemáticas relacionadas con el trabajo realizado, que necesitan ser atendidas. Considerando esta información se realiza una actualización del *backlog* (ceremonia 12) la cual, como se muestra en la figura, podría comprender regresar historias de usuario del *backlog* del *sprint* al *backlog* del proyecto (si no fueron aprobadas por el propietario del producto) o generar historias de usuario nuevas. De manera similar, tareas nuevas podrían ser agregadas al *backlog* del proyecto. Sin embargo, se debe ser cuidadoso pues añadirle más elementos podría impactar de manera negativa en la fecha de término del producto establecida. Si fuera el caso, se debe considerar reducir el alcance de este.

Hasta este punto terminaría un *sprint*, y se puede iniciar uno nuevo considerando, en general, la secuencia de ceremonias a partir de la sexta. Más *sprints* se llevan a cabo hasta que se alcanza el número establecido de ellos para el proyecto y, de manera ideal, se entrega el producto final al propietario de este.

7.3 ¿GRAN DISEÑO AL INICIO O DEUDA TÉCNICA?

El proceso *Scrum*, y por lo habitual, cualquiera asociado a un método ágil, es por naturaleza ligero y poco detallado. Por ello se debe ser cuidadoso en su adopción, ya que podría generar confusiones importantes en sus practicantes, en especial si estos tienen poca experiencia en el diseño y el desarrollo de sistemas. Por ejemplo, el valor del *Manifiesto ágil* “Software funcionando por encima de documentación extensiva” podría ser interpretado como “No documentación”. De forma similar, la ausencia tanto del rol de arquitecto como de una ceremonia explícita de diseño de arquitectura podría ser interpretada como que en *Scrum* no se hace trabajo arquitectónico. Esto sería incorrecto. Aunque los métodos ágiles promueven la comunicación cara a cara, el desarrollo incremental y las actividades de diseño “justo en tiempo” (*just in time*), no significa que las decisiones de diseño, incluidas las de arquitectura, deban ser tomadas de forma oportunista o poco deliberada.

Una percepción recurrente en el desarrollo ágil es que adoptar un enfoque *Big Design Up Front* (discutido en la sección 3.6.2), esto es, tener un diseño completo del sistema o de su arquitectura antes de comenzar con su codificación, es malo. En lugar de ello, en muchos métodos ágiles se prefiere un enfoque “diseño emergente”, lo cual significa que el diseño del sistema va a ir “emergiendo” de su implementación. De esta forma, los practicantes de *Scrum* podrían evitar dedicar mucho tiempo para tomar decisiones de diseño, incluidas las de la arquitectura, y podrían preferir tomarlas “justo a tiempo”.

En nuestra opinión, el enfoque de diseño emergente no es siempre adecuado o posible. Existen casos en los que los sistemas a construir tienen que ver con algo completamente nuevo para el equipo de desarrollo, y por ello no es tan evidente qué decisiones de diseño tomar de forma inmediata. Minimizar o ignorar la importancia de esta situación es riesgoso y podría ser causa de muchos problemas en el futuro.

En el contexto de la planeación de sistemas, el término deuda técnica es una metáfora utilizada para referirse a la “deuda” que el equipo de desarrollo adquiere, y debe asumir, al hacer omisiones durante el diseño de estos con el fin de acelerar la implementación para cumplir fechas de entrega o expectativas de los clientes. Una deuda técnica podría ser un costo alto (reflejado, por ejemplo, en salarios o tiempo requerido) para realizar cambios a un sistema a efecto de incluir una funcionalidad nueva, o bien, proveer esta misma a más usuarios sin degradar los tiempos de respuesta.

Aunque en la práctica la deuda técnica es inevitable debido a diversos factores, que van desde requerimientos cambiantes hasta la ignorancia del equipo de desarrollo respecto de que está adquiriendo una deuda de esa clase (lo cual puede ocurrir con equipos con poca experiencia), realizar acciones para minimizarla, o no hacerlas y asumir la deuda y “sus intereses”, debe ser un aspecto a evaluar en todo proyecto de desarrollo, sea ágil o no.

7.4 DESARROLLO DE ARQUITECTURA EN SCRUM

En la actualidad, y en contraste con otras actividades incluidas en el proceso *Scrum*, no hay información muy establecida sobre cómo o en qué momento se deben realizar las actividades relacionadas con el desarrollo de arquitectura. Sin embargo, a partir de la información pública de muchos practicantes de *Scrum*, así como de experiencias propias, en las siguientes secciones describiremos en el contexto de este método ágil una instancia de un enfoque de “diseño planeado incremental”, y aspectos de soporte relacionados. Este enfoque difiere del de diseño emergente descrito en la sección anterior. En este contexto usamos la palabra “planeado” para denotar que el diseño se define de manera deliberada antes de la implementación, mediante la toma de decisiones reflexionadas. Empleamos la palabra “incremental” para denotar que el diseño se define e implementa gradualmente hasta que el producto está terminado; por ello las actividades iniciales de esta etapa no requieren un diseño completo. En esta instancia hemos considerado la inclusión de algunas ceremonias adicionales en el proceso original de *Scrum*.

A continuación describimos los detalles de estas ceremonias.

Ceremonia 4. Definición de la arquitectura base

La primera ceremonia propuesta, la número 4, es una obligatoria cuyo objetivo principal radica en la toma de decisiones orientadas a definir un diseño arquitectónico base que pueda ser refinado posteriormente. Durante ella es preferible utilizar conceptos de diseño existentes en vez de “reinventar la rueda”. Sin embargo, y como lo mencionamos en el capítulo 3, lo anterior no debiera ser una limitante a la creatividad durante el diseño. Como se observa en la figura, esta ceremonia se halla fuera de lo que en *Scrum* se considera un *sprint*, por lo tanto, solo se realiza una vez.

Con el propósito de no perder agilidad se recomienda que durante esta ceremonia el equipo de desarrollo se enfoque únicamente en las siguientes tareas:

1. Seleccionar estilo (o patrón) arquitectónico o una arquitectura de referencia para la estructuración general del sistema, instanciar los elementos del patrón o arquitectura de referencia y asignarles responsabilidades a dichos elementos.

Esta tarea se debe llevar a cabo considerando el dominio de aplicación y/o el tipo de sistema a desarrollar. Por ejemplo, el patrón arquitectónico de capas es utilizado con frecuencia para estructurar sistemas *web* (Microsoft, 2009). De esta forma, es común instanciar el patrón con tres capas asignando responsabilidades a cada una de ellas como sigue: una capa permite la solicitud de los servicios, otra proporciona estos, y una más soporta la persistencia de datos que necesitan los servicios.

2. Seleccionar una estrategia de implantación para el patrón arquitectónico seleccionado.

La tarea hace referencia a una estrategia que permita la instalación y puesta en marcha de los elementos del patrón de arquitectura seleccionado. Por ejemplo, una estrategia de implantación que podría utilizarse en sistemas *web* estructurados en tres capas es el de ubicar estas como sigue: la capa que soporta la solicitud de los servicios, en un equipo tipo cliente ligero; la que proporciona los servicios solicitados, en un servidor de aplicaciones *web*, y la que soporta los servicios de persistencia de datos, en un servidor de datos.

3. Realizar una selección inicial de tecnologías para soportar la implementación del patrón arquitectónico seleccionado.

Por ejemplo, la capa que soporta los servicios de persistencia de datos podría estar implementada en *Hibernate*¹, que es un *framework* de mapeo objeto-relacional para la plataforma Java que facilita mapear entre una base de datos relacional tradicional y el modelo de objetos de un sistema.

4. Documentar la arquitectura resultante considerando las decisiones tomadas.

Se recomienda documentar aspectos lógicos (de estructuración del sistema) y físicos (de implantación). De manera ideal, la información sobre la selección tecnológica se debe añadir a estos diagramas.

5. Verificar el diseño generado.

Esta tarea se encarga de verificar que el diseño:

- a) Permite realizar las historias de usuario con mayor valor de negocio (denotado generalmente por su prioridad). Las historias a las que nos referimos aquí podrían corresponder en naturaleza a los *drivers* de requerimientos de usuario que discutimos en el capítulo 2.
- b) Atiende las restricciones de diseño identificadas. Las restricciones a las que nos referimos aquí podrían corresponder en naturaleza a los *drivers* de restricciones que discutimos en la sección 2.1.7.3.
- c) Atiende principios generales de diseño como modularidad, cohesión alta, acoplamiento bajo y simplicidad que discutimos en el capítulo 3, sección 3.3.
- d) Promueve el desarrollo paralelo e incremental del sistema.

¹ www.hibernate.org



Estas decisiones son muy similares a lo que mostramos en la iteración inicial del caso de estudio del apéndice.

Es importante notar que esta ceremonia precede intencionalmente a la de estimación de esfuerzo (ceremonia 5). En la sección 7.2.2 explicamos que esta estimación se realiza asignando puntos de historia a las historias de usuario y tareas del *backlog* del proyecto. Si bien, en el contexto de *Scrum* no se espera tener una estimación de esfuerzo exacta en los primeros *sprints*, consideramos que contar con un diseño inicial de la arquitectura podría contribuir a mejorarla. Conocer la naturaleza de los elementos arquitectónicos contribuye a efectuar estimaciones de complejidad más precisas. De manera similar, diseñar la arquitectura involucra con frecuencia la reutilización de decisiones de diseño. Tener esta información explícita permite hacer estimaciones basadas en datos de proyectos anteriores.

Ceremonia 9. Refinamiento de la arquitectura

La segunda ceremonia propuesta, la número 9, tiene como objetivo la toma de decisiones orientadas a refinar el diseño arquitectónico para completar las historias de usuario en el *backlog* del *sprint*. Nuestra estrategia para no perder la agilidad es plantear esta como opcional durante los *sprints* 1 al *n*. La decisión de realizarla o no se basa en la información disponible sobre la complejidad técnica de cada historia de usuario en el *backlog* del *sprint* y los criterios de aceptación asociados a esta (información generada en las ceremonias 5 y 8, respectivamente).

En nuestra experiencia, una historia de usuario con una medida alta de complejidad tiene asociados por lo habitual criterios de aceptación para atributos de calidad que difieren de los que comúnmente se presentan en el dominio de aplicación del sistema a desarrollar. Esto hace evidente en muchos casos un reto técnico que requiere una solución de diseño más reflexionada. Como lo discutimos antes, tomar una decisión de diseño apresurada para resolver un reto técnico representa un riesgo que podría ser minimizado al efectuar actividades de la arquitectura en ese respecto.

Si el equipo de desarrollo decide realizar esta ceremonia, se recomienda que para cada historia de usuario con alta complejidad se enfoque en las siguientes tareas:

1. Seleccionar uno o más conceptos de diseño que satisfagan los criterios de aceptación correspondientes. Si aplica, instanciar el o los conceptos, y asignar responsabilidades en los elementos correspondientes.
2. Si el concepto seleccionado requirió generar elementos nuevos en la arquitectura, actualizar la documentación actual. Actualizar, si es necesario, la información sobre la selección tecnológica.
3. Verificar el diseño generado. En específico, verificar que las decisiones de diseño tomadas no afectan en:
 - a) Realizar las historias de usuario con mayor valor de negocio.
 - b) Atender las restricciones de diseño identificadas.
 - c) Atender principios generales de diseño como modularidad, cohesión alta, acoplamiento bajo y simplicidad.
 - d) Promover el desarrollo paralelo e incremental del sistema.

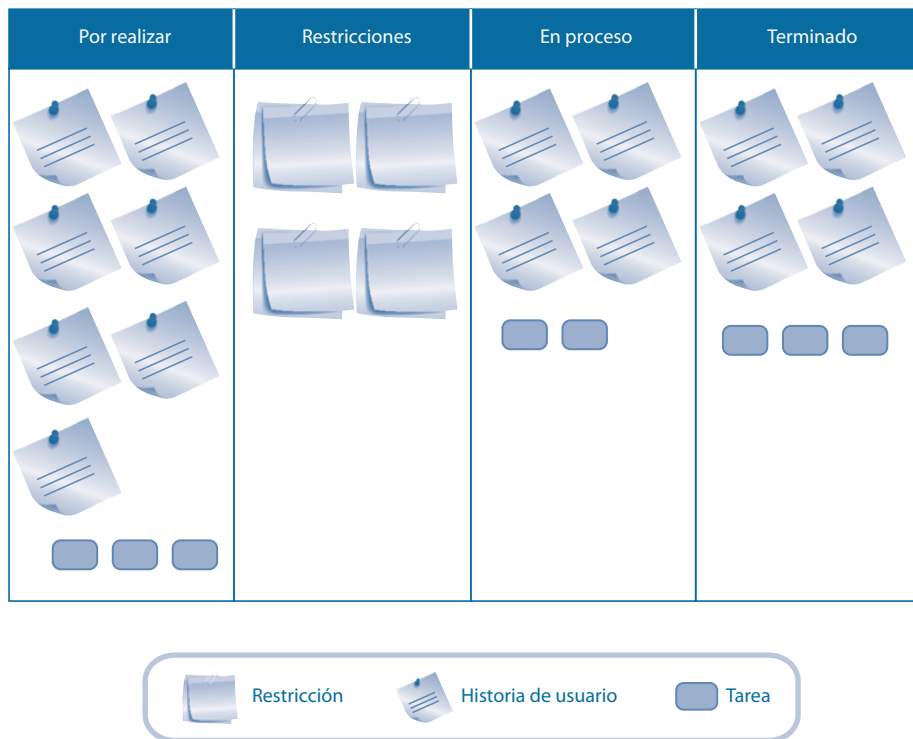
7.4.2 Especificación de atributos de calidad y restricciones

En el capítulo 2 comentamos que los atributos de calidad junto con las restricciones son los requerimientos que tienen mayor influencia sobre el diseño de la *arquitectura de software*. Esto aplica para arquitecturas diseñadas en un contexto ágil o no; por ello se hace uso de esta información en las ceremonias propuestas anteriormente. En esta sección damos algunas recomendaciones sobre cómo documentar esta información y utilizarla para beneficio de otras ceremonias, además de las propuestas para el diseño de la arquitectura.

Durante la ceremonia de especificación de historias de usuario (ceremonia 1) es muy importante que el maestro *Scrum* identifique información sobre atributos de calidad y restricciones asociadas al producto de *software*. Sin embargo, para no perder agilidad, solo debería documentar la información relevante para la crea-

ción de la arquitectura. Mencionamos anteriormente que esta noción de relevancia se asocia al hecho de que esos atributos y restricciones representen un reto técnico para el equipo de desarrollo durante las actividades de implementación.

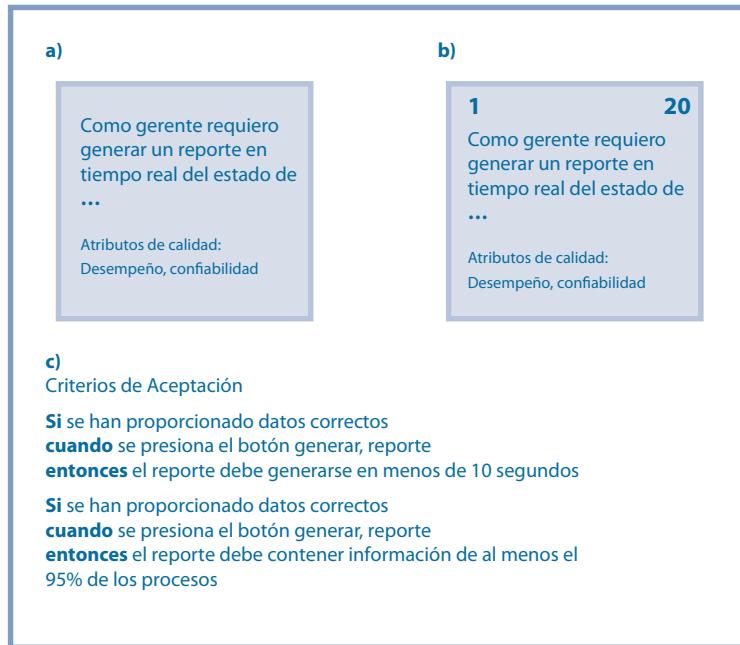
Como explicamos en el capítulo 2, las restricciones describen aspectos que limitan las decisiones que se pueden tomar para diseñar y desarrollar el sistema; incluyen con frecuencia requerimientos sobre el uso de productos de *software* y de *hardware*, métodos de diseño o implementación, o lenguajes de programación específicos. Es importante no solo documentar esta información, sino mantenerla siempre visible para todo el equipo de desarrollo, pues las restricciones aplican al sistema entero. El uso del tablero Kanban es una práctica frecuente en *Scrum* para mantener a la vista el estado actual del proyecto respecto del trabajo terminado, el trabajo en proceso y el trabajo por realizar (Cohn, 2009). En la figura 7-4 se presenta el boceto de un tablero Kanban con una columna para las restricciones como medio para mantenerlas observables a los miembros del equipo de *Scrum*.



© Humberto Cervantes Maceda / Perla Velasco Elizondo / Luis Castro Caraga. Ilustraciones: © Gamegix / Shutterstock.

› **Figura 7-4.** Tablero Kanban con información sobre restricciones.

Por otra parte, la información sobre atributos de calidad podría documentarse en las historias de usuario correspondientes escribiendo en la parte inferior el nombre o nombres del atributo a manera de recordatorio. Durante la ceremonia 5 puede utilizarse esta información como recordatorio de que la historia de usuario involucra un reto técnico que requiere una solución de diseño más reflexionada. De esta forma, la medida de complejidad asignada a la historia debería reflejar esta situación, como se muestra en la figura 7-5 b. Durante la ceremonia 7 tanto el recordatorio de atributo de calidad como la medida de complejidad de la historia de usuario son de utilidad para el equipo de desarrollo para soportar la definición de las pruebas relacionadas a tales atributos, lo cual se ilustra en la figura 7-5 c.



»Figura 7-5. Propuesta para documentar atributos de calidad en historias de usuario.

7.4.3 ¿Vistas de arquitectura?

En el capítulo 4 explicamos que, en los métodos de desarrollo tradicionales, las *vistas* son artefactos comunes para documentar la arquitectura de un sistema. Una vista describe una o más estructuras arquitectónicas en términos de los elementos que las conforman. Una vista se conforma, en términos generales, por: 1) un diagrama en donde se representan los elementos de la estructura, y 2) información textual que ayuda a la comprensión de dicho diagrama.

En un contexto ágil la generación de vistas no es una actividad carente de valor. Por el contrario, como lo hemos discutido antes, disponer de documentación apropiada de la arquitectura facilitará la comprensión por parte del equipo de desarrollo de qué partes tiene el sistema que van a construir y cuáles son las decisiones de diseño relevantes a la construcción de estas partes. Sin embargo, en un contexto ágil es importante cuidar que la documentación arquitectónica contenga el nivel necesario de información para soportar el *sprint* en curso y evitar incluir aquella que no aporta valor en el desarrollo de dicho *sprint*.

Tomando en cuenta la forma de trabajo de los equipos ágiles, es preferible adoptar un enfoque de documentación simple y efectivo. Por ejemplo, es recomendable optar por el uso de notaciones informales para elaborar los diagramas. De manera similar, se sugiere que los diagramas correspondientes a la arquitectura base (generados en la ceremonia 4), al igual que artefactos como los *backlogs*, estén siempre visibles para el equipo. Con ello, durante el desarrollo del sistema los diagramas pueden utilizarse como puntos de referencia en las discusiones sobre las decisiones de diseño específicas que deben aplicarse para el refinamiento de la arquitectura durante los *sprints*.

En un entorno ágil la documentación debería evolucionar a la par de la arquitectura y el desarrollo del sistema. Sin embargo, según nuestra experiencia, no siempre es necesario que los bocetos resultantes del refinamiento arquitectónico (generados en la ceremonia 9) estén siempre visibles para el equipo que desarrolla. Sin embargo, sí son artefactos de utilidad durante las retrospectivas y, en algunos casos, en las juntas diarias de *sprint*.

7.4.4 El arquitecto de *software*

Ya hemos explicado antes que el arquitecto de *software* es el responsable de definir, documentar, mantener y vigilar la implementación del diseño de la arquitectura. De manera ideal, en el contexto de *Scrum* estas responsabilidades deberían ser compartidas por varios miembros del equipo de desarrollo. Sin embargo, puede darse el caso en que una sola persona asuma este rol. De ser así, ella podría fungir como asesor o facilitador en materia de desarrollo de arquitectura, dejando al equipo las principales responsabilidades sobre esta.

Es importante notar que en un contexto como el de *Scrum* todo miembro del equipo que realiza actividades de arquitectura debería ser capaz también de llevar a cabo otras funciones propias de este rol. Con esto queremos decir que, por ejemplo, un arquitecto que no codifica es contradictorio y, en consecuencia, poco común en el contexto de *Scrum*.



EN RESUMEN

En este capítulo abordamos el tema de los métodos ágiles. Describimos el proceso que sigue *Scrum*, el cual es un método ágil muy popular actualmente, y discutimos cómo actividades del ciclo de desarrollo de la arquitectura pueden realizarse en este proceso.

Planteamos en específico el rol de arquitecto de *software* y la forma en que la arquitectura se desarrolla en *Scrum*. En este método ágil, este rol se comparte por lo habitual por varios miembros del equipo de desarrollo. Cuando se asume por un solo miembro, este funge únicamente como asesor o facilitador y deja al resto del equipo las principales responsabilidades sobre la misma.

Respecto de la forma en que en *Scrum* se desarrolla la arquitectura describimos un proceso, adaptado a partir del utilizado comúnmente, el cual define dos ceremonias orientadas a tal desarrollo.

PREGUNTAS PARA ANÁLISIS

1. En el contexto de *Scrum* para el desarrollo de sistemas de *software*, discuta con un(a) compañero(a) la relevancia de contar con una arquitectura y su documentación para: i) mejorar la comunicación al interior del equipo de desarrollo; ii) preservar la información sobre la arquitectura; iii) guiar la codificación del sistema, y iv) proveer un lenguaje común entre este equipo, el propietario del producto y el maestro *Scrum*.
2. Hay una frase popular que dice algo así: "...usted no necesita un diseño para construir una casa para un perro, pero es mejor tener un poco un diseño si usted va a construir un rascacielos". En el contexto del desarrollo ágil, discuta con un(a) compañero(a) cuándo esta frase es apropiada para el diseño de arquitectura. Justifique su respuesta.
3. Considere que, utilizando *Scrum*, una compañía de desarrollo de sistemas va a crear uno de análisis automático de comentarios escritos en las redes sociales por los clientes de una cadena de restaurantes. ¿Qué decisiones de diseño deberían tomarse antes de iniciar el primer *sprint*?, ¿cuáles se deberían documentar y comunicar al equipo de desarrollo? y ¿cuáles serían las consecuencias de no hacerlo?