

# Introducción a la Arquitectura de Software

Maximiliano Cristiá

Ingeniería de Software

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

Agosto de 2008

## 1. Breve introducción histórica

En los primeros años de la construcción de software no existía el diseño del sistema como una etapa independiente de la programación. Pensar la descomposición del programa antes de programar se comienza a proponer, investigar y aplicar a principios de la década del 70 cuando se comienza a distinguir entre pequeños y grandes programas o sistemas [1]. Precisamente uno de los pioneros en esté área fue David L. Parnas con sus trabajos seminales en ocultación de información y desarrollo de familias de programas [2, 3, 4], seguido por el trabajo de Liskov y Guttag referido al diseño basado en tipos abstractos de datos [5, 6, 7]. Más o menos en la misma época pero con un impacto en la industria mucho mayor Tom DeMarco, Edward Yourdon, Larry Constantine y Glenford Myers desarrollaron y consolidaron el análisis y diseño estructurado [8, 9, 10].

Recién a principios de la década del 90 algunos investigadores, principalmente ligados a la Carnegie-Mellon University y al Software Engineering Institute, comienzan a ver la necesidad de investigar y desarrollar un nivel de abstracción superior al del diseño, al que llamaron Arquitectura de Software. Este trabajo se condensa principalmente en el libro pionero de Mary Shaw y David Garlan [11] y en el volumen más elaborado de Len Bass, Paul Clements y Rick Kazman [12]. Precisamente la segunda edición de este último [13] junto a otro volumen más especializado producido también en el SEI [14] y al trabajo de un grupo de ingenieros de software de Siemens [15] son tal vez las expresiones más modernas y acabadas de la especialidad.

El contenido de este apunte se basa en [11, 13] más algunas contribuciones personales del autor.

## 2. Arquitectura y diseño en el ciclo de vida del sistema

Siguiendo el modelo de desarrollo conocido como Modelo de Cascada, la arquitectura y el diseño de un sistema se conciben en la fase denominada, precisamente, Arquitectura del Sistema la cual es la que le sigue a la Ingeniería de Requerimientos. Es decir que según este modelo, para poder definir la arquitectura y el diseño de un sistema es necesario contar con todos los requerimientos del sistema. Si bien contar con la totalidad de los requerimientos es claramente una gran ventaja para poder definir una buena arquitectura, no es lo que ocurre en la realidad. Como hemos analizado al estudiar el diseño de software, lo usual es que se deba comenzar a definir el diseño cuando se cuenta con un conjunto incompleto de requerimientos y que, peor aun, el concepto de "conjunto completo de requerimientos" es utópico ya que jamás se contará con todos los requerimientos puesto que estos van apareciendo durante toda la vida del sistema. Precisamente, vimos que el diseño debía definirse pensando tanto en los requerimientos que se tienen como en los que faltan y en los que vendrán. Exactamente lo mismo aplica a la arquitectura del sistema.

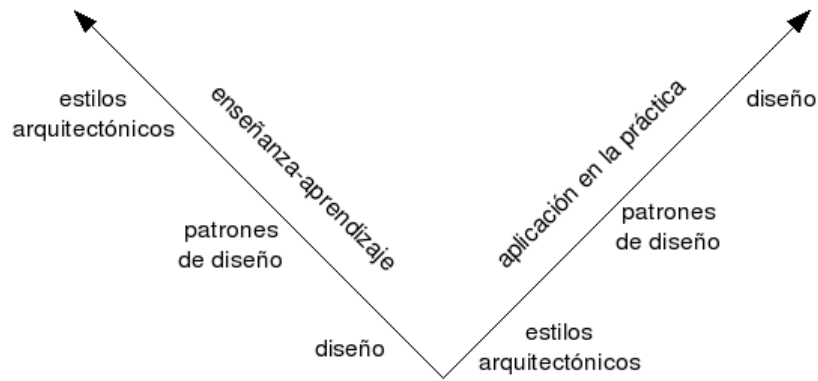


Figura 1: Las etapas de la fase Arquitectura del Sistema se enseñan-aprenden de forma opuesta a la forma en que se aplican en la práctica.

Consideramos que la fase de Arquitectura del Sistema puede subdividirse en tres etapas como se muestra en la Figura 1:

1. Elección del estilo arquitectónico
2. Selección de los patrones de diseño
3. Diseño de componentes

Notar, por otra parte, que el proceso de enseñanza-aprendizaje de todo lo relativo a la fase Arquitectura del Sistema discurre en el sentido opuesto al que se aplica en la práctica como se intenta graficar en la Figura 1. En otras palabras, se enseña/aprende primero a diseñar componentes, luego se enseña/aprende a utilizar patrones de diseño y finalmente se enseña/aprende lo relativo a estilos arquitectónicos; mientras que un ingeniero de software en su práctica cotidiana primero debe elegir el estilo arquitectónico, luego definir cuáles patrones de diseño aplicará y finalmente diseñar los módulos del sistema.

### 3. El ciclo de la arquitectura

En esta sección veremos cómo la arquitectura del sistema afecta a su entorno y este también afecta a la arquitectura. El material de esta sección es un resumen del capítulo 1 de [13].

Para poder desarrollar este tema es necesario proponer una primera definición de arquitectura de software.

**Definición 1 (Arquitectura de Software)** *La arquitectura de software de un programa o sistema de cómputo es la estructura o estructuras del sistema que comprenden elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos.*

Dado que en el capítulo sobre Diseño de Software también se hizo referencia al concepto de estructura pero con un significado levemente diferente, conviene clarificar este punto antes de continuar. En aquel apunte "estructura" se utiliza como sinónimo de *vista*. Una vista es la representación de un conjunto coherente de elementos arquitectónicos y sus relaciones; en este sentido una vista es un documento que describe parte de la arquitectura del sistema. En la definición 1, estructura se utiliza para designar el "conjunto coherente de elementos arquitectónicos y sus relaciones" en sí y no la documentación de ese conjunto. La diferencia es sutil pero significativa: cuando hablamos de

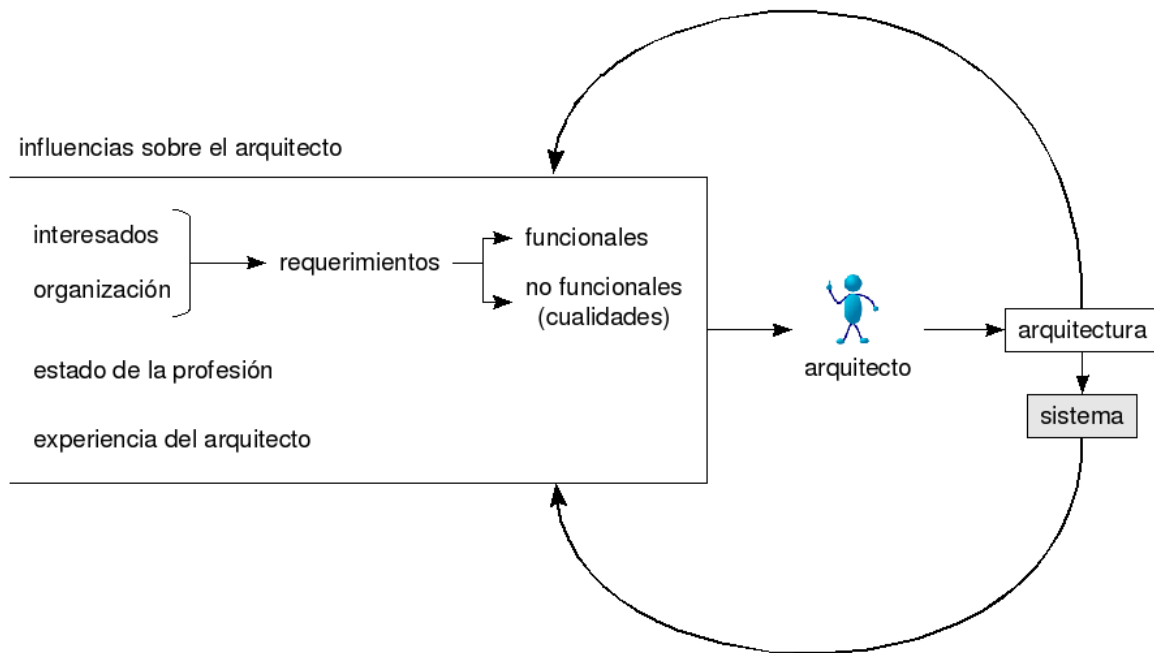


Figura 2: Influencias mutuas entre la arquitectura y su entorno.

estructura, hablamos de elementos de software tal y como están en el sistema, en el código o en ejecución; en tanto que cuando hablamos de vista, nos referimos a la documentación de esos elementos. Claramente, la arquitectura del sistema es el conjunto de estructuras, que no necesariamente coincide con la documentación (vistas) que supuestamente le corresponden.

Habiendo aclarado esos conceptos, observemos que la definición 1 es similar a la definición de diseño de software dada en el capítulo sobre diseño, aunque no es exactamente igual. Ahondaremos en estas diferencias en la sección 4.

Antes de analizar las influencias mutuas entre la arquitectura y su entorno, es conveniente introducir el concepto de *interesado* o *involucrado* en el sistema<sup>1</sup>. Los interesados en una arquitectura de software de un cierto sistema son todas aquellas personas u organizaciones que tienen alguna injerencia o interés en el sistema. Un listado incompleto es el siguiente: programadores, administradores, testers, usuarios finales, dueño del sistema, organizaciones con las cuales el sistema interactúa, bancos que financian la construcción del sistema, etc. Nunca se repetirá la cantidad de veces suficiente lo importante que es determinar con la mejor precisión posible el conjunto de interesados en el sistema antes de comenzar a delinear su arquitectura. Determinar el conjunto de interesados en el sistema es responsabilidad o injerencia de los ingenieros a cargo de la Ingeniería de Requerimientos, como primer paso antes de comenzar con la captura de los requerimientos. Tanto los ingenieros de requerimientos como el equipo responsable de la arquitectura del sistema, más tarde o más temprano, deberán interactuar con todos y cada uno de los interesados para validar diferentes aspectos del sistema. Un interesado relevante que no sea consultado tempranamente sobre la arquitectura del sistema, será una fuente de problemas en el futuro.

Las influencias mutuas entre la arquitectura de un sistema y su entorno se denominan *Architecture Business Cycle* o ABC y se condensan en la Figura 2. La arquitectura de software de un sistema es el resultado de combinar decisiones técnicas, sociales y del negocio. Los ingenieros de software deben convencerse que su trabajo está influenciado por el negocio y el entorno social que los rodea. Es ilusorio e ingenuo suponer que uno podrá determinar la arquitectura únicamente basándose en consideraciones técnicas. Los interesados, algunos de ellos más que otros, más tarde o más temprano

<sup>1</sup>Ambos términos son nuestra traducción para *stakeholder*.

presionarán al arquitecto del sistema para que la arquitectura tenga tal o cual característica que no necesariamente es la mejor desde el punto de vista técnico. Por ejemplo, el gremio de empleados estatales puede presionar para que un sistema de administración impositiva preserve una interfaz con el usuario en modo texto dado que la mayor parte de los usuarios finales (es decir los afiliados al gremio) que domina esta tecnología es renuente a capacitarse o adaptarse a interfaces más modernas. Algo semejante ocurre con las influencias provenientes de la organización que desarrolla el sistema. Por ejemplo, si la mayor parte de los programadores tiene cierta fluidez en Java y toda su tecnología, diversas gerencias y grupos dentro de la organización presionarán explícita o implícitamente para que la arquitectura se oriente hacia esa tecnología. Los conocimientos técnicos del arquitecto así como también el estado del arte de la práctica profesional de la Ingeniería de Software también son fuertes condicionantes no técnicos a la hora de definir la arquitectura de un sistema. Si el arquitecto domina el DOO entonces sus arquitecturas tenderán a seguir esa técnica; si se está ideando un sistema de comercio electrónico, la industria y la cultura técnica imperantes presionan para utilizar webservices independientemente de que sea la tecnología óptima para el sistema en cuestión.

El ciclo ABC se cierra pues una arquitectura exitosa tenderá a convertirse en la referencia obligada dentro de la organización para estructurar sistemas semejantes o elaborar líneas de productos. Asimismo los usuarios finales o los clientes serán renuentes a definir una nueva arquitectura para un nuevo sistema si la anterior fue la base para un sistema que les brindó buenas prestaciones. Finalmente, ciertos sistemas basados en ciertas arquitecturas logran rebasar las fronteras de una organización y se convierten en estándares de-facto o referencias obligadas de la industria por lo que terminan teniendo una influencia importante sobre la práctica profesional (tal es el caso, por ejemplo, de la tecnología Web la cual fue pensada por Tim Berners-Lee para compartir los resultados de la investigación dentro del Laboratorio Europeo de Física de Partículas (CERN) de Ginebra pero que terminó convirtiéndose un una tecnología de alcance universal).

### 3.1. La influencia de los requerimientos no funcionales en la definición de la arquitectura de un sistema

En la Figura 2 se presenta a los requerimientos como una influencia determinante en la concepción de la arquitectura de software del sistema. Más aun los requerimientos se dividen en dos grandes clases: funcionales y no funcionales (también llamados cualidades del sistema). Normalmente los requerimientos funcionales no ocupan solo la primera posición sino la única a la hora de definir la arquitectura del sistema. Si bien hasta cierto punto esto puede ser razonable suele ser una fuente inagotable de problemas. Usualmente requerimientos no funcionales o cualidades tales como modificabilidad, seguridad, desempeño, tolerancia a fallas, testeabilidad, etc. no son tenidas en cuenta ni por los arquitectos del sistema ni por la mayoría o todos los interesados. En general la necesidad de que el sistema verifique algunas de estas cualidades se percibe recién cuando este entra en producción o en las etapas finales del desarrollo.

El problema de esta situación radica en que por lo general lograr que un sistema casi terminado tenga un desempeño superior, sea modificable, seguro o tolerante a fallas (ni que hablar de que verifique varias de estas cualidades simultáneamente) es casi imposible si no se pensó desde su concepción, es decir si esas cualidades no se incorporaron conscientemente en la arquitectura del sistema. No ocurre lo mismo, en la mayoría de los casos, con el requerimiento de agregar nuevas funcionalidades; suele ser costoso pero en general no requiere rehacer el sistema por completo. Los requerimientos funcionales por lo general son *discretos* en el sentido de que agregando o modificando algunas líneas de código en unos pocos lugares es suficiente para implementarlos, mientras que los requerimientos no funcionales son, por lo común, *continuos* en el sentido de que es necesario agregar o modificar código en todas partes para implementarlos. En consecuencia no prever un requerimiento no funcional suele ser mucho más costoso que no tener en cuenta un requisito funcional.

Actualmente se considera que los requerimientos no funcionales deben guiar la definición de la

arquitectura del sistema tanto como los funcionales. Observar que esto fue notado tempranamente por Parnas al proponer que los cambios probables del sistema (es decir la modificabilidad) deben guiar la definición de los módulos y sus interfaces, en tanto que los cambios probables son consecuencia, entre otras cosas, de los requerimientos actuales, del dominio de aplicación, etc.

## 4. La diferencia entre arquitectura y diseño

En la sección 1 decíamos que recién a principios de la década de los 90, la comunidad de profesionales relacionados con la producción de software se comenzó a plantear la necesidad de expresar la estructura del sistema en un nivel de abstracción superior al del diseño. Sin embargo, la definición 1 no da pistas sobre cuál es ese otro nivel estructural, ni cómo expresar la estructura de un sistema en ese nivel, ni cuál es la diferencia entre diseño y arquitectura. Por este motivo proponemos una definición alternativa de arquitectura de software.

**Definición 2 (Nivel Arquitectónico)** *El nivel arquitectónico de la estructura de un sistema es aquella descripción donde se utilizan conectores diferentes a llamada a procedimiento y/o se imponen restricciones importantes entre los componentes y/o aparecen distintos tipos de componentes en la descripción.*

Antes de analizar la definición clarificaremos algunos conceptos que en ella aparecen:

**Componente.** Entidad computacional activa que implementa algún requisito funcional o parte de este.

**Conector.** Mecanismo que mediatiza la comunicación, coordinación o cooperación entre componentes.

**Tipo componente.** Componentes que comparten características estructurales, en particular, y fundamentalmente, los mismos tipos de interfaz.

**Tipo interfaz.** Forma de interacción con el entorno semántica y estructuralmente única.

Hechas estas aclaraciones analizaremos brevemente la definición propuesta. En primer lugar notar que la definición no habla de la arquitectura del sistema sino de una *descripción* particular de la arquitectura, por lo que decimos que esta definición complementa a la dada inicialmente. Observar que la definición refiere a las mismas actividades que se llevan a cabo en el diseño solo que en un nivel de abstracción diferente. En un sentido, la arquitectura *es* diseño.

La diferencia distintiva con respecto al nivel del diseño radica en que según nuestra definición la descripción del nivel arquitectónico implica el uso de elementos de software que no tienen una representación directa en la mayoría de los lenguajes de programación; es decir, elementos abstractos con los cuales trabaja el arquitecto y que los programadores deberán refinar y proyectar sobre la tecnología de implementación disponible. En el nivel arquitectónico (no del diseño) se debe hacer hincapié en componentes con interfaces complejas y/o con relaciones complejas entre ellos y, primordialmente, conectores semánticamente más ricos. En este sentido la definición 2 sigue las ideas planteadas originalmente por Shaw y Garlan [11] en cuanto a que en el nivel arquitectónico los conectores son tratados (o deberían ser tratados) como elementos de primer nivel y no como subalternos de los componentes, reducidos a llamada a procedimiento o memoria compartida. Este es, sin dudas, uno de los aportes fundamentales que separan a la arquitectura del diseño según se lo concebía tradicionalmente. Considerar a los conectores como elementos de primer nivel implica entender que una buena arquitectura depende tanto de las relaciones entre sus componentes como de los componentes en sí. Más aun, dado que los conectores en el nivel arquitectónico son abstracciones no directamente

representables en el lenguaje de programación, suele ser muy común que estos se conviertan en componentes al pasar al nivel del diseño. En otras palabras, un conector arquitectónico puede ser tan complejo que se requiera un buen número de líneas de código lo que implica que a nivel de diseño esas líneas de código deben ser asignadas a elementos que no implementan requisitos funcionales y deben ser diseñados.

Observar que el DOO no entra dentro del nivel arquitectónico pues solo utiliza llamada a procedimiento, no hay restricciones entre sus componentes (cualquier objeto puede invocar servicios de cualquier otro) y solo se utiliza un tipo de componente (los objetos). Esta observación coincide con lo dicho en el capítulo de diseño en el sentido que el DOO es una forma de diseño de relativo bajo nivel de abstracción.

Una vez descrita la arquitectura se pasa al nivel del diseño, diseñando (es decir descomponiendo en partes). Por eso, en un sentido, la arquitectura *es* diseño.

También podemos determinar si una decisión estructural es arquitectónica o no por medio del Criterio de Localía, el cual dice que una decisión estructural no es de diseño (y por lo tanto es arquitectónica) sí y sólo sí un sistema que verifica esa decisión puede ser *extendido* a un sistema que no la verifica. Por ejemplo, estructurar un sistema siguiendo el estilo cliente/servidor es una decisión arquitectónica puesto que el sistema puede ser extendido a un sistema *peer-to-peer* que no verifica el estilo. Por otro lado, si ese mismo sistema utiliza, por ejemplo, el patrón de diseño Strategy, al extenderlo a un sistema *peer-to-peer* seguirá utilizándolo, por lo que el uso del patrón es una decisión de diseño (y no arquitectónica).

Sin embargo, no siempre es simple diferenciar claramente entre el nivel de diseño y el nivel arquitectónico. En última instancia es el ingeniero el que traza la línea que divide ambos niveles en cada proyecto. Precisamente, no disponemos aun de una notación que permita expresar siempre el nivel arquitectónico sin adentrarse en el detalle del diseño. Sin embargo, ciertas estructuras permiten describir este nivel, o son más útiles en ese nivel que en el nivel del diseño. Una buena aproximación para describir la arquitectura sin describir el diseño yace en el concepto de *estilos arquitectónicos* que desarrollamos en siguiente sección.

## 5. Estilos arquitectónicos

Los estilos arquitectónicos son una generalización y abstracción de los patrones de diseño.

**Definición 3 (Estilo Arquitectónico)** *Caracteriza una familia de sistemas que están relacionados por compartir propiedades estructurales y funcionales.*

*También puede definirse como la descripción de los tipos componente y de los patrones de interacción entre ellos.*

Notar que, a diferencia de los patrones de diseño, la definición apunta a describir sistemas completos y no partes de sistemas. Nadie supone que podrá describir un sistema completo mediante el patrón de diseño Composite o Abstract Factory o Command, pero sí puede hacerlo mediante un estilo arquitectónico.

Otra forma de caracterizar un estilo arquitectónico es respondiendo las siguientes preguntas, entre otras:

- ¿Cuál es el vocabulario de diseño, es decir los tipos de componentes y conectores?
- ¿Cuáles son los patrones estructurales permitidos?
- ¿Cuál es el modelo computacional subyacente?
- ¿Cuáles son los invariantes esenciales?

- ¿Cuáles son algunos ejemplos comunes de su uso?
- ¿Cuáles son las ventajas y desventajas?
- ¿Cuáles son las variantes (especializaciones y deformaciones) comunes?
- ¿Cuál es la metodología de desarrollo?

El resto de este capítulo trata sobre la descripción y aplicación de seis estilos arquitectónicos.

## Referencias

- [1] F. DeRemer and H. Cron, “Programming-in-the-large versus programming-in-the-small,” *IEEE Transactions on Software Engineering*, vol. 2, no. 2, pp. 80–86, 1976.
- [2] D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.
- [3] ———, “Designing software for ease of extension and contraction,” *IEEE Transactions on Software Engineering*, vol. 5, no. 2, pp. 128–137, Mar. 1979.
- [4] D. L. Parnas, P. C. Clements, and D. M. Weiss, “The modular structure of complex systems,” *IEEE Transactions on Software Engineering*, vol. 11, no. 3, pp. 259–266, 1985.
- [5] B. Liskov and S. N. Zilles, “Programming with abstract data types,” *SIGPLAN Notices*, vol. 9, no. 4, pp. 50–60, 1974.
- [6] J. Guttag, “Abstract data type and the development of data structures,” *Communications of the ACM*, vol. 20, no. 6, pp. 396–404, 1977.
- [7] B. Liskov and J. Guttag, *Abstraction and Specification in Program Development*. Cambridge, MA, USA: The MIT Press, 1986.
- [8] T. DeMarco, *Structured Analysis and System Specification*. New York, NY, USA: Yourdon Press, 1978.
- [9] G. J. Myers, *Composite/Structured Design*. New York, NY, USA: Van Nostrand Reinhold, 1978.
- [10] E. Yourdon and L. Constantine, *Structured Design*. Englewood Cliff, NJ, USA: Prentice-Hall, 1979.
- [11] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Upper Saddle River: Prentice Hall, 1996.
- [12] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Reading: Addison-Wesley, 1998.
- [13] ———, *Software architecture in practice (second edition)*. Boston: Pearson Education, 2003.
- [14] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.
- [15] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stad, *Pattern-Oriented Software Architecture — A System of Patterns*. John Wiley Press, 1996.