

## Arquitectura del Software: arte y oficio

*La arquitectura de un software puede entenderse como aquella estructura del programa que cohesiona las funcionalidades más críticas y relevantes (necesarias para el sistema), y que sirve de soporte al resto de funcionalidades finales (necesarias para el usuario). Su especificación es ampliamente aceptada como el problema central de diseño de un sistema de software complejo. Uno de los principios de las metodologías modernas de desarrollo de software es priorizar la definición, el diseño, la implementación y la evaluación de la arquitectura del software. La esencia de este principio es dedicar los mínimos esfuerzos a implementar un prototipo estable de arquitectura que garantice la viabilidad del proyecto en las fechas más tempranas posibles. Este artículo reflexiona sobre la importancia de priorizar la arquitectura tanto para el producto de software como para el proceso de desarrollo, y sobre los beneficios potenciales que esta práctica puede reportar.*

### Introducción

Casi todos hemos observado alguna vez la construcción de un edificio. Comienza por los cimientos, luego las columnas y vigas, las distintas plantas, hasta tener un esqueleto de soporte. Después se construyen paredes, suelos, puertas y ventanas, instalaciones eléctricas y de fontanería, bancadas, etc. Basta un mínimo de sentido común para ni siquiera imaginar la posibilidad de levantar una pared antes que las columnas. En resumen, primero se crea la estructura o esqueleto del edificio, y luego se ensamblan las distintas partes. La primera sirve de soporte a las segundas, que aportan la mayoría de las funcionalidades básicas del inmueble. ¿Qué pasaría si se cometen errores en una u otra? Si se olvida construir una columna y se detecta al finalizar el edificio, probablemente este no obtenga nunca la cédula de habitabilidad. Sin embargo, si lo que se olvida es instalar una bañera o un armario empotrado se pierde solo una funcionalidad del inmueble (para el usuario final), pero indudablemente será habitable y dicho fallo será probablemente subsanable con un esfuerzo relativamente pequeño.

Esta forma de proceder es una estrategia general de solución de problemas en multitud de disciplinas sociales y técnicas. Una de ellas es la creación de software. A diferencia de la construcción de un edificio "común", el software no se rige por leyes físicas ni por procedimientos conocidos, sino que es inherentemente específico y experimental. Como indica su nombre, la característica principal del software es ser *soft*, es decir, flexible, elástico y, en general, muy específico para la solución de una tarea concreta, sobre sistemas de hardware concretos, por personal con actitudes, aptitudes, formación y experiencia concretas. Todo esto hace del diseño de un software particular una tarea generalmente única, creativa, con las incertidumbres y riesgos que ello conlleva.

Uno de los principios de las metodologías modernas de desarrollo de software es priorizar la definición, el diseño, la implementación y la evaluación de la arquitectura del software, que es como se conoce al esqueleto o estructura del sistema. Desde el punto de vista de qué debe hacer el software, la arquitectura se define a partir de un conjunto de requisitos críticos funcionales, de rendimiento, o de calidad. Considerando cómo el software debe dar solución a tales objetivos, la arquitectura constituye el problema central de diseño, es decir, el conjunto de estructuras, clases y atributos principales del software y sus interfaces de comunicación. Desde otro punto de vista más tangible, la arquitectura se materializa en el conjunto de componentes de código fuente y ejecutables que implementan

**Priorizar la arquitectura aporta beneficios al proceso de construcción del software.**

dicho esqueleto, lo que posibilita demostrar y evaluar en qué medida el diseño da solución a aquellos requisitos críticos.

Dado que no existe una teoría establecida sobre cómo proceder, el diseño, implementación y evaluación de la arquitectura de un software complejo puede realizarse a través de un proceso iterativo de prototipado, demostración y corrección. Este proceso permite atender y resolver en los inicios del proyecto los riesgos asociados a los requisitos más críticos y a las decisiones de diseño más difíciles, que son aquellos que más pueden comprometer el éxito del proyecto. Así, el equipo de desarrollo debe diseñar, construir y estabilizar primero la arquitectura del software antes de diseñar e implementar el conjunto de componentes elementales que se integran en la arquitectura y que aportan las funcionalidades finales de usuarios.

La esencia del principio de priorizar la arquitectura es dedicar los mínimos esfuerzos a garantizar la corrección de las partes más importantes, costosas e indefinidas del sistema, y cuyo prototipo permita una demostración tangible de la viabilidad del proyecto.

### Descripción de la arquitectura del Software

El software tradicional, como único proceso ejecutándose en un único ordenador, no supone arquitecturas complejas ni grandes riesgos. Sin embargo, los sistemas actuales aprovechan componentes comerciales, código abierto, sistemas distribuidos, nuevos y diversos lenguajes de programación, entornos de alojamiento (hosting) y de ejecución remota, y otras dependencias externas, que convierten a la arquitectura de un sistema en su producto técnico más crítico.

Alcanzar una arquitectura estable que dé garantías sobre la viabilidad del proyecto, se considera el punto de transición entre lo que se suele denominar la fase de ingeniería

## Arquitectura del Software: arte y oficio

(definición del producto y su solución) y la fase de producción (construcción, integración, evaluación y entrega del producto). En la primera se toman las decisiones más importantes mientras que en la segunda se realizan los mayores gastos y esfuerzos.

La clave del éxito y a su vez la dificultad del principio de priorizar la arquitectura consiste en definir qué es y qué no es la arquitectura. Si incluimos demasiados detalles perdemos la propiedad de configuración mínima necesaria (o más simple posible), que nos permite demostrar, con el mínimo esfuerzo y en fechas tempranas, la corrección de la solución diseñada. Si, por el contrario, definimos una configuración más simple de lo necesario, estaremos probablemente ignorando requisitos, riesgos, o interacciones críticas, lo cual impide dar garantías sobre el éxito del proyecto antes de realizar los mayores esfuerzos y gastos de la fase de producción.

Desde un punto de vista de gestión del proceso de desarrollo, podemos identificar dos dimensiones para manejar la arquitectura:

- descripción de arquitectura: subconjunto del modelo de diseño del software. Incluye elementos significativos de la arquitectura y excluye el diseño de las componentes básicas. Resuelve decisiones sobre qué desarrollar, qué reutilizar y qué comprar. Contiene notación ad hoc (textos y gráficos) necesaria para comprender los modelos. Debe incluir también los criterios de evaluación de la arquitectura.
- versión estable de arquitectura: subconjunto suficiente de componentes ejecutables que permiten demostrar lo antes posible que el método de solución (diseño, tecnología, tiempo y costes estimados) puede resolver satisfactoriamente la definición del producto (objetivos, alcance, rendimiento, beneficios, calidad).

Desde una perspectiva técnica, como se ha comentado antes, la arquitectura engloba requisitos críticos, decisiones de diseño, componentes de código fuente, y componentes ejecutables, información que debe ser modelada e incluida en el documento de descripción de arquitectura. Este contexto de información puede ser representado a través de las siguientes vistas y diagramas UML:

- vista de casos de uso: describe los casos de uso críticos de la arquitectura; puede ser modelado estáticamente a través del diagrama de casos de uso;
- vista de diseño: describe los componentes del modelo de diseño significativos para la arquitectura, es decir, aquellos que aportan la estructura y funcionalidad básica del sistema; puede ser modelado estáticamente mediante diagramas de clases y objetos;
- vista de proceso: describe interacciones en tiempo de ejecución de los componentes de la arquitectura en un entorno distribuido, incluyendo distribución lógica de procesos, hilos de control, comunicación entre procesos; puede ser modelado estáticamente a través del diagrama de distribución (*deployment diagram*);
- vista de componente: describe los componentes del conjunto de implementación (código fuente) significativos

para la arquitectura desde la perspectiva de los programadores; puede incluir componentes de prueba, comerciales, de simulación, que luego pueden o no ser considerados en la vista de entrega; puede ser modelado estáticamente a través del diagrama de componente.

- vista de entrega: describe los componentes ejecutables de la arquitectura, incluyendo la correspondencia entre procesos lógicos y recursos físicos del entorno de explotación; puede ser modelado estáticamente a través del diagrama de distribución.

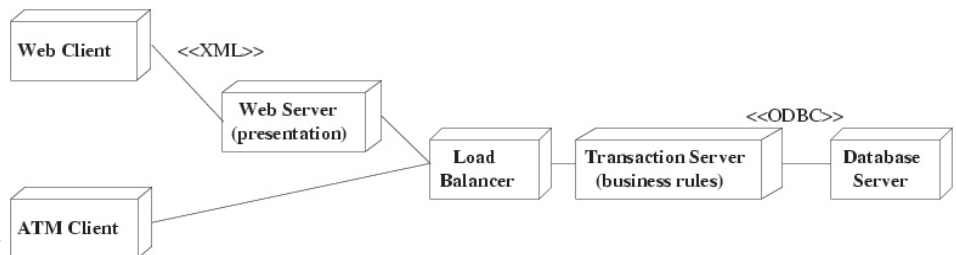


Figura 1: Ejemplo de diagrama de distribución (UML) de una arquitectura cliente-servidor. Muestra la relación entre los componentes críticos de mayor alcance ignorando detalles de bajo nivel del diseño. Fuente: Kazman et al., CMU/SEI-2004-TR-011, July 2004.

Las vistas de casos de uso y diseño son generalmente necesarias en cualquier sistema, mientras que las tres restantes dependen de la complejidad de su arquitectura. Todas las vistas pueden ser modeladas dinámicamente mediante los diagramas UML de comportamiento, es decir, los diagramas de secuencia, colaboración y actividad. Como se ha comentado antes, estas decisiones y diagramas deben incluirse, argumentarse y relacionarse en el documento de descripción de la arquitectura. A modo de referencia sobre qué aspectos considerar en la descripción de la arquitectura, la organización IEEE define el estándar IEEE Std 1471-2000 para tales efectos (ver [http://standards.ieee.org/reading/ieee/std\\_public/description/se/1471-2000\\_desc.html](http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html)).

### Beneficios potenciales del principio de priorizar la arquitectura

La estrategia de priorizar la arquitectura aporta significativos beneficios en materia de corrección al proceso de construcción del software, entre ellos:

- evaluación de la solución (diseño + implementación) mediante demostraciones tangibles de sus capacidades desde fases muy tempranas de desarrollo;
- atención temprana a riesgos relacionados con la arquitectura que, generalmente, coinciden con aquellos que pueden conducir a mayores daños;
- propicia la construcción incremental del software (integración temprana) y las correspondientes actividades de verificación;
- propicia el desarrollo orientado a demostraciones periódicas de productos funcionales;
- interfaces correctas permiten una cooperación eficiente entre diseñadores e implementadores.

## Arquitectura del Software: arte y oficio

En las siguientes secciones explicaremos con más detalle estos beneficios.

### 1. Evaluación basada en demostración

La razón principal que justifica priorizar la arquitectura es demostrar lo más pronto posible que la solución (el diseño) es correcta para resolver el objetivo (los requisitos). Como ya se comentó, la arquitectura es punto de transición entre la fase de ingeniería, donde se toman las decisiones más importantes, y la fase de producción, donde se aquellas decisiones se traducen en los mayores gastos del proyecto. En la primera intervienen usualmente un máximo de 2 ó 3 personas, quienes idean la solución y toman las decisiones críticas. En la segunda fase, un equipo de desarrolladores de tamaño variable en función de la complejidad del proyecto, colabora en la implementación de la solución durante un período de tiempo mayor.

La arquitectura es la materialización temprana, grosera y de mínimo coste de las decisiones más importantes. El prototipo ejecutable de arquitectura debe permitir demostrar que la solución ya es madura, es decir, que tales decisiones son efectivas para resolver los principales casos de uso del software que se va a construir antes de realizar los gastos ingentes asociados a la fase de producción, en la cual un equipo de desarrolladores implementará e integrará la mayor parte del código durante un período de tiempo casi siempre superior al de la fase de ingeniería. La arquitectura debe dar garantías de que la solución diseñada es realizable dentro de las restricciones de tiempo, personal, y presupuestos, o sea, que el proyecto es viable.

### 2. Atención a riesgos relacionados con la arquitectura

Otro principio esencial de los métodos actuales de desarrollo de software es la gestión de riesgos relacionados con el proceso de desarrollo o con el producto de software. Este principio significa identificar al inicio del proyecto las dudas e incertidumbres que existen sobre qué hacer o cómo hacerlo, y definir planes para investigarlas y resolverlas.

Si la arquitectura constituye los cimientos del software a construir, los riesgos relacionados con esta son lógicamente los más críticos del proyecto, es decir, aquellos que pueden producir los mayores errores o daños. Centrarse rápidamente en el diseño, implementación y estabilización de un prototipo de arquitectura implica tener que gestionar necesariamente dichos riesgos y resolverlos. Así estaremos eliminando las mayores causas potenciales de errores graves.

### 3. Integración temprana

Un tercer beneficio de priorizar la arquitectura es que propicia una estrategia incremental de construcción del

software como forma de aplicación del principio “divide y vencerás”.

Una arquitectura estable puede concebirse como una estructura de soporte del software, en la que se puedan ir integrando gradualmente las implementaciones de las diferentes funcionalidades finales. Cada nueva funcionalidad implementada e integrada es probada como unidad y como parte del todo en el que se inserta. Es decir, un nuevo requisito recién implementado puede ser inmediatamente validado desde la perspectiva del usuario en el contexto de la aplicación en el que se usará finalmente.

La integración temprana permite dosificar los esfuerzos de realización de tests de integración. Al integrar gradualmente cada funcionalidad en un prototipo de arquitectura previamente verificado y validado, la comprobación de la nueva funcionalidad dentro del prototipo se simplifica notablemente por reducirse las fuentes probables de error. Se deberá comprobar también que se mantiene la corrección de todo el prototipo en su interdependencia con la nueva funcionalidad.

### 4. Demostraciones periódicas a clientes y usuarios

Este procedimiento incremental de construcción garantiza que en cada instante del proceso de construcción del software exista un prototipo funcional validado, aunque parcialmente terminado. Así, cada prototipo parcial permitiría realizar demostraciones periódicas a clientes/usuarios finales con el propósito de detectar, tan pronto como aparezcan, desviaciones entre lo que estos esperan y el software implementado. Además, puede planificarse la construcción de forma que se priorice la integración de aquellas funcionalidades más útiles a clientes/usuarios, para crear versiones parciales usables por ellos. Esta forma de hacer facilita por tanto la visibilidad del progreso, es decir, el seguimiento y revisión de planes a través de la evaluación precisa de los estados parciales del producto. Existen otras dos ventajas adicionales de esta práctica. Por un lado, refuerza la moral del equipo de desarrollo al presenciar avances tangibles y frecuentes. Por otro lado, mejora sensiblemente la satisfacción y compromiso de clientes/usuarios al apreciar el desarrollo gradual de su producto, al sentirse partícipes del mismo, y al contar desde fechas tempranas con versiones usables, aunque incompletas.

### 5. Cooperación eficiente a partir de interfaces de módulos

La interfaz de un módulo de software se refiere a la forma en la que dicho módulo interactúa con el resto del sistema: cómo se identifica, cómo se activa, qué datos necesita, y qué resultados devuelve. La especificación correcta de las interfaces de los módulos de la arquitectura propicia la cooperación y el paralelismo en la implementación de estas partes. El responsable de cada parte observará el resto de las partes como cajas negras disponibles con las que sabrá cómo interactuar. En general dichas partes

## Arquitectura del Software: arte y oficio

podrán desarrollarse en paralelo e integrarse posteriormente.

### Conclusiones

Uno de los principios de las metodologías modernas de desarrollo de software es priorizar la definición, el diseño, la implementación y la evaluación de la arquitectura del software, que es como se conoce al esqueleto de soporte del sistema. La arquitectura implementa los requisitos más críticos a través de las estructuras de programa de mayor alcance en el sistema. Por ello la arquitectura encierra los mayores riesgos del desarrollo. La clave del éxito y, a su vez, la dificultad del principio de priorizar la arquitectura consiste en definir qué es y qué no es la arquitectura. Sus componentes deben ser los suficientes para garantizar la viabilidad del proyecto y, a su vez, los mínimos que permitan dar tales garantías con el mínimo gasto de tiempo, esfuerzo y recursos en general. Alcanzar una arquitectura estable reporta significativos beneficios al proceso de construcción del software, entre ellos la construcción incremental del software, la evaluación frecuente mediante demostraciones periódicas, la resolución de los riesgos más peligrosos en etapas tempranas, el aumento de la satisfacción y compromiso de clientes y usuarios,

Algunas referencias recomendables para ampliar información sobre la arquitectura del software:

<http://www.sei.cmu.edu/architecture/definitions.html>  
<http://www.sei.cmu.edu/publications/documents/04-reports/04tr011.html>  
<http://www-306.ibm.com/software/rational/uml/>  
[http://en.wikipedia.org/wiki/Software\\_architecture](http://en.wikipedia.org/wiki/Software_architecture)  
<http://www.bredemeyer.com/whatis.htm>  
<http://www.softwarearchitectureportal.org/WICSA/>

el mantener alta la moral del equipo de desarrollo, y una cooperación más efectiva entre sus miembros. A efectos prácticos, todo proceso de desarrollo de software, y en particular las metodologías de diseño y construcción, deben definirse a partir del reconocimiento de este protagonismo de la arquitectura dentro del producto de software y dentro del propio proceso.

Autor: Ramón Mollineda  
 Más información:  
[otri@iti.upv.es](mailto:otri@iti.upv.es)

### INSCRIPCIÓN ABIERTA

### II JORNADAS DE TESTEO DE SOFTWARE

Una oportunidad para conocer las nuevas técnicas, modelos y metodologías que sirven para mejorar la calidad de nuestro software.

<http://www.iti.upv.es/JTS2005>  
[jts2005@iti.upv.es](mailto:jts2005@iti.upv.es)

21 y 22 de Abril de 2005  
 Salón de Actos  
 Centro de Desarrollo  
 de Turismo  
 Pº de la Alameda, 37  
 Valencia

10% DESCUENTO PARA  
 INSCRIPCIONES ANTES DEL 4  
 DE MARZO

#### Colaboran:

COMPCWARE 

inQA.labs  
 Software Testing 

Borland®

 UNIVERSIDAD  
 POLITÉCNICA  
 DE VALENCIA

Las II Jornadas sobre Testeo de Software, organizadas por el Instituto Tecnológico de Informática, contarán con la participación de expertos internacionales en el área de software testing así como con profesionales interesados en debatir y compartir experiencias sobre los cambios que se están produciendo en la calidad del software.

#### Objetivo:

Los beneficios que aporta un buen sistema de testeo de software incluyen el cumplimiento de plazos, la disminución de errores en el desarrollo e implantación, la reducción de costes y la mejora en la satisfacción del cliente.

La jornada pretende explicar la importancia y los fundamentos del testeo de software y describir los procesos básicos que cada compañía moderna de software debe implementar para garantizar cierto nivel de calidad en sus productos de software. Finalmente, la jornada se dirige a la presentación y explicación de técnicas, modelos y metodologías que se pueden utilizar para llevar a cabo un proceso básico de testeo.

#### Seminario complementario:

Con el fin de facilitar el aprovechamiento de las jornadas a todos los asistentes, se impartirá en forma independiente un seminario previo que tendrá por finalidad introducir los conceptos básicos del testeo. El seminario tendrá lugar el día 20 de abril de 16:00 a 20:00, en las instalaciones del ITI (Ciudad Politécnica de la Innovación, Edif. 8G, UPV - Camino de Vera S/N, Valencia).