Desarrollo de Software

Programa de la asignatura:
Diseño y arquitectura de software

Unidad 1. Arquitectura de software

Clave:

Licenciatura: TSU:

15142424 / 16142525

Universidad Abierta y a Distancia de México

DCEIT

México, Ciudad de México, enero del 2026

Índice

INTRODUCCIÓN

En esta primera unidad revisarás los principios de la arquitectura de software así como su importancia en el diseño; así mismo, conocerás las distintas propuestas existentes en la industria del desarrollo del software y en la academia acerca de un lenguaje formal, llamado lenguaje descriptor de arquitectura, e identificarás cuántas y cuáles son las vistas de la arquitectura y cómo se pueden conjuntar estas vistas a través del lenguaje UML, de manera que sea habitual para cualquier usuario que tenga acceso a la descripción de la arquitectura del software a construir.

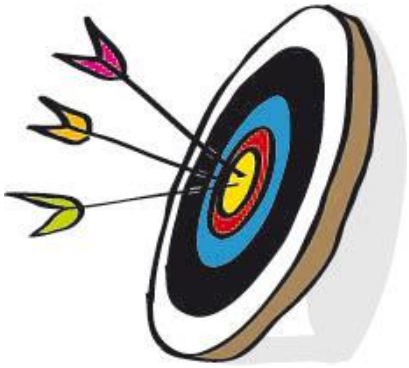
Las vistas te llevarán a tener una descripción de modelo de forma clara con un enfoque arquitectónico, señalando los principales patrones existentes aplicables a la estructuración de la arquitectura de software, así como para ubicarla dentro del ciclo de vida del desarrollo, primordialmente en la fase de diseño.

Adicionalmente, exploraremos diversas arquitecturas de software que han surgido como respuestas innovadoras a los desafíos contemporáneos en el desarrollo de sistemas. Cada una de estas arquitecturas presenta enfoques específicos para abordar distintos aspectos, desde la distribución eficiente de servicios hasta la implementación de tecnologías avanzadas como la inteligencia artificial.

LOGROS

Al término de esta unidad lograrás:

- Comprender el concepto de arquitectura de software y su importancia en el proceso de diseño.
- Identificar los lenguajes de descripción de arquitectura, y su impacto en el éxito de un proyecto de desarrollo de software.
- Representar una vista, mediante el modelo de vistas 4+1 y un lenguaje formal.



- Identificar el impacto de la arquitectura de software en la fase de diseño de un proyecto de software.

COMPETENCIA ESPECÍFICA

- Analiza las herramientas de arquitectura de software para utilizarlas en el proceso de diseño de una aplicación, propiciando la implementación de vistas, UML y lenguajes descriptores de arquitectura.

1.1. INTRODUCCIÓN A LA ARQUITECTURA DE SOFTWARE

La arquitectura de software es una disciplina emergente que con sus herramientas ayuda a abstraer, conceptualizar y modelar de manera óptima estilos y patrones para aplicarlos en la construcción de software. Podrás observar el papel primordial que juega esta disciplina para el éxito o fracaso de un proyecto de software.

Figura1. Arquitectura de software
Fuente: http://pushpaktechnologies.com/development.html

Figura2. Evolución del desarrollo de software

También, es importante aclarar que la fusión entre los conceptos que se abordarán sobre Arquitectura de software (AS) y la práctica será un tema que involucre mucha dedicación, porque influyen muchos factores para abarcar esta disciplina, no consolidada del todo, que brinda las herramientas, principios, recomendaciones y recomendaciones para la obtención de un diseño arquitectónico, base para el desarrollo de un sistema de software. En el contexto de la arquitectura de software, se suele mencionar el enfoque industrial y teórico; lo cual se puede referir al diseño documentado o sobre papel y al sistema de software ya desarrollado y puesto en marcha.

Cuando se diseña un sistema es necesario realizar ajustes, por ejemplo, consideraciones del cliente, cambios en la tecnología, etcétera, pero es recomendable siempre seguir los métodos establecidos para cumplir con las etapas del desarrollo.

Para profundizar sobre el concepto de arquitectura de software, revisa el documento U1. Definición de arquitectura de software, en la sección Material de apoyo de la unidad.

1.1.1. Origen y objetivo de la arquitectura de software

En comparación con otras actividades que realiza el ser humano, el desarrollo de software es una disciplina joven; algunos dirán que se puede contar entre 60 u 80 años de su aparición formal.

La evolución del desarrollo de software ha avanzado a pasos acelerados, ya que en la actualidad son muy pocos los ambientes o entornos donde no involucra de una manera u otra el funcionamiento del software, y hay casos en donde la dependencia se ha hecho total, llegando a tal grado que la vida humana, la seguridad de un país o la automatización del mecanismo de apagado de la cafetera, dependen del correcto funcionamiento del software.

La acelerada evolución ha hecho que el desarrollo de software se deba apoyar en otras disciplinas como la ingeniería y la arquitectura, en su sentido más puro, para así tomar lo mejor de ellas en partes donde ya se tiene mucha experiencia. Se ha pasado por un proceso de aprendizaje y los resultados están probados en la mayoría de los casos como exitosos. No es de extrañarse que la base de la arquitectura de software se ha tomado de la industria de la construcción, utilizando sustantivos comunes como arquitecto y arquitectura, que toman su definición de ese ámbito para aplicarlos de la misma manera en el campo del desarrollo de software.

Haciendo un breve recorrido histórico sobre cómo se ha aplicado esta evolución, e identificando una semejanza entre la construcción de estructuras tangibles (edificios, puentes, pirámides, aeropuertos, entre otros) y la construcción de software, se podría decir que si se dibuja una línea de tiempo imaginaria, se ha ido desde vivir en cuevas, pasando por el adobe, hasta llegar al ladrillo y la prefabricación de estructuras utilizables en cualquier ámbito que se requiera. Los mismos principios se aplican, sin mover muchos elementos, en la construcción de software.

En esta analogía es importante aclarar que la descripción de esta línea de tiempo imaginaria sólo será útil para estructuras pequeñas, como una casa-habitación o incluso un edificio pequeño de cinco u ocho niveles. Cuando se trata de construir edificios de más de 100 niveles, capaces de soportar grandes presiones a causa de la fuerza del viento, terremotos, o el cálculo y equilibrio de las cargas máximas que soporta el material de construcción utilizado para mantener la seguridad estructural, o si se trata de la construcción de un puente que cruza de

una ciudad a otra por encima de un enorme río o incluso el mar, entonces se está hablando de cosas y requisitos de diseño totalmente distintos entre ambos tipos de estructuras descritas.

En la construcción del software se ha evolucionado desde la construcción artesanal, enfocada hacia problemas específicos y de operaciones repetitivas (tales como simples impresiones de mensajes a las salidas estándar), pasando por el ámbito estrictamente académico como el cálculo de ecuaciones de cientos o miles de variables, el número PI, software que simula modelos para predecir el clima, realizar el cálculo de impuestos, definir la dirección de naves espaciales, hasta el empleo completo de complejas arquitecturas para poder soportar plataformas de software que prestan servicios inimaginables para la mayoría de la población y solventan los problemas cotidianos como la escritura de una carta en un procesador de palabras, la creación de complejas redes sociales virtuales y reales, o el cobro automático del derecho a abordar un vehículo de transporte público con la simple presentación frente a un lector de una tarjeta dotada de chip.

La adaptabilidad de un sistema de software con otros semejantes u otras tecnologías emergentes, su robustez, facilidad de mantenimiento e incluso gran parte del éxito de sus usuarios al consumir sus servicios dependerá de un diseño correcto de la arquitectura que lo soporta.

1.1.2. Lenguajes descriptores

Cuando los requerimientos del software a desarrollar han salido de la etapa del análisis y han sido entregados al arquitecto, éste decide que es tiempo de pensar **en cómo** hará que esos requerimientos queden cubiertos con una sólida arquitectura de software*; entonces, deberá modelar* esas características de los requerimientos en la mencionada arquitectura, es decir, **describirla** con una convención gráfica o un lenguaje formal de alto nivel propio para representar la arquitectura resultante.

Tras una larga tradición de construcción de arquitecturas, se ha creído de manera generalizada que para definir una arquitectura estándar es posible y necesario tener a disposición un ambiente gráfico abundante, como es el caso del modelado CASE o UML, y así también es necesario que la visualización del sistema sea con componentes gráficos. El mismo arquitecto, previamente mencionado, puede añadir o poner estos componentes según sea la necesidad que se presente, sin tener que aprender un lenguaje formal que describa de mejor manera dichas propuestas.

Se ha dicho que la evolución del software ha llegado a grandes soluciones basadas en las mejores prácticas de otras disciplinas, y podría pensarse que esta misma evolución ya tendría disponible para los arquitectos de software un estándar totalmente aceptado en la industria del desarrollo de software sobre cómo plasmar en forma gráfica la arquitectura, sin embargo, no lo hay.

En la actualidad, con el auge que tienen las tecnologías web y móviles, podría suponerse que se cuenta con todos los elementos para poder hacer una arquitectura que cumpla con lo antes mencionado, o más aún, con servicios web o ambientes heterogéneos; se ha pensado en el mundo ideal, pero al ver la realidad de un arquitecto de software se observa que la situación es mucho más compleja y que las cosas no son tan sencillas como pudiera apreciarse.

Como ejemplo de la situación compleja podemos observar la acelerada aparición de tecnologías móviles y de nuevas propuestas de arquitectura ha hecho que para modelar un servicio web, el acceso desde o hacia un dispositivo móvil o la infraestructura en la nube, se empleen técnicas altamente cuestionables o incluso totalmente fuera de estándares, sea cual sea para poder dar cumplimiento a ello (por mencionar una de tantas situaciones); la realidad es que existe un desfase con relación al avance y lanzamiento de nuevas tecnologías y dispositivos.

Para poder describir una arquitectura de manera estándar y adecuada surgió el uso de los lenguajes de descripción de arquitecturas (ADL), los cuales se utilizan para satisfacer requerimientos descriptivos que necesitan un alto nivel de abstracción, requisito que puede

cumplir, por ejemplo, UML. Este nivel de abstracción en los sistemas se debe a que cada vez se necesita que el software resuelva más problemas de la vida cotidiana, y se ha permeado a todas las ramas y fases de la vida de las personas, no en ambientes tan controlados como la academia, con las fórmulas matemáticas.

Siguiendo con la comprensión de los ADL es necesario considerar una definición un poco más formal, la que se habrá de aplicar desde ahora es la de un lenguaje descriptivo de modelado, del cual su principal interés es la estructura de alto nivel del producto de software antes que los detalles finos de desarrollo e implementación de los módulos que lo conforman. Ante esta situación, se podrían enumerar los ADL que se encuentran actualmente en la industria para dar una idea de cuántos esfuerzos se han hecho por las empresas u organismos por generar su propio estándar, algunos con resultados bastante buenos y otros han tenido grandes dificultades para ser siquiera aceptados.

De esta forma, en la que cada quien lanza y usa su propia definición y especificación de los ADL, se puede observar que no existe una definición generalizada de los lenguajes, pero comúnmente debe pensarse que estos lenguajes proporcionan un modelo explícito de componentes, conectores y sus respectivas configuraciones y, en el marco de la presente asignatura, se define como aplicación (Reynoso, 2004, p. 4).

Los **lenguajes de interconexión** de módulos son los predecesores de los ADL, pero éstos comienzan un desarrollo serio a partir de principios de los noventas, un poco antes de que la arquitectura de software se tomara como una propuesta seria y necesaria. Los ADL cuentan con cuatro criterios que los definen como una entidad: componentes, conectores, configuraciones y restricciones. Para poder considerar un lenguaje que pertenezca a la familia de los ADL debe soportar por lo menos los siguientes elementos:

- Componentes
- Conexiones
- Composición jerárquica

- Paradigmas de computación
- Paradigmas de comunicación
- Modelos formales subyacentes
- Soporte de herramientas para modelado, análisis, validación y verificación
- Composición automática de código aplicativo
- Abstracción de componentes
- Relatividad
- Capacidad para modelar componentes
- Tipos y verificación de tipos

Esta lista es tomada de diversos autores de forma sintética.

A continuación, a modo de resumen, se presenta la siguiente tabla para suponer el entendimiento de los requisitos que debe cumplir un ADL.

Componentes	Conectores	Configuraciones arquitectónicas	Soporte de herramientas
Interfaz	Interfaz	Comprensibilidad	Especificación activa
Tipos	Tipos	Composicionalidad	Múltiples vistas
Semántica	Semántica	Heterogeneidad	Análisis
Restricciones	Restricciones	Restricciones	Refinamiento
Evolución	Evolución	Refinamiento y trazabilidad	Generación de código
Propiedades no funcionales	Propiedades no funcionales	Escalabilidad	Dinamismo
		Evolución Dinamismo Propiedades no funcionales	

Tabla 1. Requisitos de un ADL

Considerando las propuestas señaladas, a continuación, se definen los elementos constitutivos primarios que, más allá de la diversidad existente, son comunes a la ontología de todos los ADL, y habrán de ser orientadores de su tratamiento en este estudio (Reynoso, 2004):

Elementos	Definición	Algunos ejemplos más comunes
Componentes (descripciones de caja—y-línea)	Elementos computacionales primarios de un sistema. Se exponen en diferentes interfaces, que definen la interacción de un componente y entorno.	Clientes, base de datos, servidores, filtros, objetos, pizarras.
Conectores (descripciones de caja—y-línea)	Interacciones entre componentes. Se exponen en diferentes interfaces, que definen los roles de los componentes que participan en la interacción.	Tuberías (pipes), protocolos, cliente-servidor, broadcast de eventos.
Configuraciones o sistemas	Compuestos como grafos de componentes y conectores. La topología del sistema se define aparte de los componentes y conectores en los ADL más avanzados. Los sistemas pueden ser jerárquicos.	----------
Propiedades.	Representan información semántica sobre un sistema más allá de su estructura. Diferentes ADL dan importancia a las diferentes clases de propiedades, pero definen las no funcionales o admiten herramientas complementarias para su análisis.	Troughput y la latencia probables, cuestiones de seguridad, dependencias de bibliotecas o configuraciones mínimas de hardware y tolerancia a fallas.

Restricciones	Representan condiciones de diseño de acuerdo con las evoluciones en el tiempo. Una restricción común seria las configuraciones topológicas admisibles.	El número de clientes que se conectan al mismo tiempo a un servicio.
Estilos	Representan: familias de sistemas, vocabulario de tipos de elementos de diseño y reglas para componerlos. Algunos prescriben un framework o patrones arquitectónicos.	Arquitecturas de flujo de datos basados en tuberías y filtros, sistemas en capas.
Evolución	No todos los ADL soportan los procesos de evolución para el refinamiento de sus rasgos, son pocos los que lo soportan dependiendo de lenguajes que ya no son los de diseño arquitectónico, sino los de programación.	----
Propiedades no funcionales	Son necesarias para simular la conducta **runtime** analizar la conducta de los componentes, imponer restricciones y mapear implementaciones sobre procesadores determinados.	------

Tabla 2. Elementos constitutivos primarios comunes a la ontología de todos los ADL

A continuación, se te invita a revisar el siguiente tema, Vistas de la arquitectura. No olvides comentar a tu Docente en línea las dudas que tengas durante tu estudio en la unidad.

1.1.3. Vistas de la arquitectura

En la AS no todo está homogenizado ni es igual para todo, en cuestiones de vistas de arquitectura de software, así también la opinión de académicos y organismos reconocidos como ISO, IEEE,ACM, IBM, por mencionar algunos, no han presentado alguna definición de las vistas. Por ello, las recomendaciones de los frameworks, o marcos de trabajo, se usan típicamente para basarse en las vistas y se usan con mucho respeto, pero igual con algo de recelo, por parte de

los involucrados; estos mismos marcos de trabajo y modelos arquitectónicos acostumbran a ordenar las diferentes perspectivas de una arquitectura en términos de vistas.

Entonces se dice que las vistas de una arquitectura son un "subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado" (Reynoso, 2004, p.) para aplicarlo en la solución al problema presentado por los requerimientos del cliente. De manera más concreta, las vistas de una arquitectura dan el sentido de las partes que conformarán la aplicación; como la mejor manera de dividir las operaciones que realizará la solución de software, facilitando su entendimiento y mantenimiento.

De una manera más coloquial, imagina que tuvieras la necesidad de cocinar un pastel. Las vistas de este pastel serán:

- Ingredientes y preparación de la cobertura.
- Ingredientes y preparación del interior del pastel.
- Recipiente de presentación/almacenamiento.

Es decir, cómo se ve, cómo se hace y cómo se hará uso de él; no se debe confundir con los pasos para su elaboración. Existen muchas opiniones de las vistas de arquitectura. Hay comités muy reputados junto con sus propuestas como RM-ODP, RUP, RDS, MDA, MOF, MEMO, XMI o IEEE 1471-2000.

No se hace un análisis exhaustivo de las vistas que propone cada uno de estos comités, solo se mencionará que las vistas de una arquitectura **describen, de manera lógica y de sentido común, la división del trabajo total** que hará la aplicación de software. Esta descripción es muy cercana en cuanto a sus pasos a cualquier metodología de desarrollo de software.

La vista más simple que se puede presentar es (***Microsoft***):

- Lógica
- Conceptual
- Física

Estos tres puntos mencionados describen de manera simple, pero relevante, lo que una arquitectura denota en su vista para alcanzar la solución conceptualizada por el arquitecto.

Por lo tanto, una vista es, para definirla, "un subconjunto resultante de practicar una selección o abstracción sobre una realidad desde un punto de vista determinado" (Reynoso, 2004):

Zachman (Niveles)	**TOGAF (Arquitecturas)**	**4+1 (Vistas)**	**[BRJ99] (Vistas)**	**POSA (Vistas)**	**Microsoft (Vistas)**
Scope	Negocios	Lógica	Diseño	Lógica	Lógica
Empresa	Datos	Proceso	Proceso	Proceso	Conceptual
Sistema Lógico	Aplicación	Física	Implementación	Física	Física
Tecnología	Tecnología	Desarrollo	Despliegue	Desarrollo	
Representación		Casos de uso	Casos de uso		
Funcionamiento					

Tabla 3. Vistas en los marcos de referencia (Reynoso, 2004)

Conjunto típico de las vistas de una arquitectura

Con el conocimiento conceptual de qué es una vista, que se revisó en el subtema "Vistas de la arquitectura", ahora se precisará mejor el significado arquitectónico de las vistas y para qué proporcionar un amplio conocimiento sobre estas.

La expresión **múltiples vistas** significa que la solución de software tiene dentro de sí muchas partes independientes, pero colaborativas que la conforman, esta representa lo más importante para la ingeniería de software y los requerimientos de sistemas.

Las razones para ello son variadas: en primer lugar, no existe una fundamentación coherente para su uso en la disciplina; en segundo término, muchos estudiosos las consideran problemáticas, porque la existencia de múltiples vistas introduce problemas de integración y consistencia entre ellas. Sin embargo, los arquitectos practicantes las usan de todas maneras, porque simplifican la visualización de sistemas complejos (Reynoso, 2004).

La conceptualización y la abstracción de un sistema (de cualquier tipo) no son originarias de la AS, sino que se trata de una teoría más amplia que se aplica y se conoce como análisis sistémico. La necesidad de dividir el trabajo resultante de cualquier desarrollo de software es muy simple: que cada parte haga el trabajo que le corresponda sin entrometerse en la participación del otro, aunque tengan (o deban tener) comunicación.

Existe un problema común en AS, que consiste en tratar de solventar un problema cuya complejidad sea muy alta por el elevado número de reglas de negocio que debe atender con el desarrollo de software que esté conformado por una sola pieza única e indivisible, provocando que sea intratable e insostenible, por ello el arquitecto es el encargado de modelar soluciones fiables, propone dividir las complejidades a diferentes vistas para dividir la citada complejidad y hacer el acercamiento hacia la solución más fácil para él y para quienes estarán participando en etapas posteriores del ciclo de vida del desarrollo del software.

Si se pudieran dividir las vistas de la AS se podría obtener dos versiones, según el ámbito de aplicación: el normal y el referido desde UML.

A continuación, se presenta la relación de ambas versiones.

Área	Vista	Diagrama UML	Concepto principal
Estructural	Vista estática	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interfaz
	Vista de casos de uso	Diagrama de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	Vista de implementación	Diagrama de componentes	Componente, interfaz, dependencia, realización
	Vista de despliegue	Diagrama de despliegue	Nodo, componente, dependencia, localización

Dinámica	Vista de máquinas de estados	Diagrama de estados	Estado, evento, transición, acción
	Vista de actividad	Diagrama de actividad	Estado, actividad, transición de terminación, división, unión
	Vista de interacción	Diagrama de secuencia	Interacción, objeto, mensaje, activación
		Diagrama de colaboración	Colaboración, interacción, rol de colaboración, mensaje
Gestión del modelo	Vista de gestión del modelo	Diagrama de clases	Paquete, subsistema, modelo

Tabla 4. Versiones de las vistas de la AS según el ámbito de aplicación, normal y referido desde UML

La propuesta de vistas no tiene límites en cuanto cantidad y orden refiriéndose a la aplicación de ellas a la solución de cualquier problema.

Podría dedicarse mucho tiempo a discutir el concepto y la conformación de las diferentes vistas, y de hecho los numerosos simposios que ha habido sobre la manera en cómo deben usarse las vistas de una arquitectura han sido de interés, mas no han definido conclusiones sobre el método correcto ni estándar de hacerlo.

Se acostumbra poner las vistas que denotan "niveles de abstracción" en cuadros superpuestos, por ejemplo, como si fueran análogas a las capas del modelo OSI, pero ¿son ambas representaciones estrictamente homólogas?, ¿constituye el conjunto de las vistas un sistema?, ¿por qué cada vez que se enumeran los artefactos característicos de cada vista aparece la palabra "etcétera" o la expresión "elementos principales? (Reynoso, 2004).

Estos niveles de abstracción (vistas) sí pueden considerarse como una representación homóloga del modelo OSI, pero no en su sentido más estricto, pues ambas denotan un orden jerárquico

de superposición donde los niveles inferiores soportan los superiores y sin la existencia de ellos no se pueden siquiera pensar en la colocación de los niveles de orden superior. Además, este conjunto de vistas sólo puede considerar sistema si hay cooperación entre las distintas vistas para llegar a un fin (dictado por los requerimientos dados por el cliente).

1.1.4. Modelo Vistas 4+1

Modelo vinculado al RUP (Rational Unified Process) y definido en 1995 por Philippe Kruchten en el cual se definen cuatros vistas para la arquitectura de software. El uso de estas múltiples vistas permite abordar la arquitectura por separado desde el punto de vista de cada participante en el sistema: usuario final, desarrollador, administrador del proyecto, etc. y administrar los requisitos tanto los funcionales y como los no funcionales de manera separada. Cada una de las vistas definidas mostrará la arquitectura de software que se documenta, pero cada una debe presentarse de forma diferente mostrando aspectos distintos del sistema.

Vista Lógica: Comprende las abstracciones fundamentales del sistema a partir del dominio de problemas. Se utiliza el estilo orientado a objetos y la base para el diseño principal que se sigue es “mantener un modelo de objetos simple y coherente a través de todo el sistema”. Se apoya principalmente de los requisitos funcionales –lo que el sistema debe brindar en términos de servicio a los usuarios.

Vista de Proceso: Toma en cuenta requerimientos no funcionales, tal como el rendimiento y la disponibilidad del sistema. Describe los aspectos de concurrencia y sincronización del diseño. En esta vista se incluyen los procesos que existen en el sistema, así como la forma en que éstos se comunican. Representa los procesos de negocio, sus flujos de trabajo y los elementos operacionales que componen el sistema.

Vista de Desarrollo: Describe la organización estática del software en su ambiente de desarrollo. Es una vista desde la perspectiva del programador. Describe la organización de los

diversos módulos de software en el ambiente de desarrollo. El software es "empacado" en subsistemas que pueden ser desarrollados por uno o varios programadores. En esta vista se toman en cuenta los requerimientos internos relacionados con la facilidad para el desarrollo, administración del sistema, re-uso, restricciones, etc.

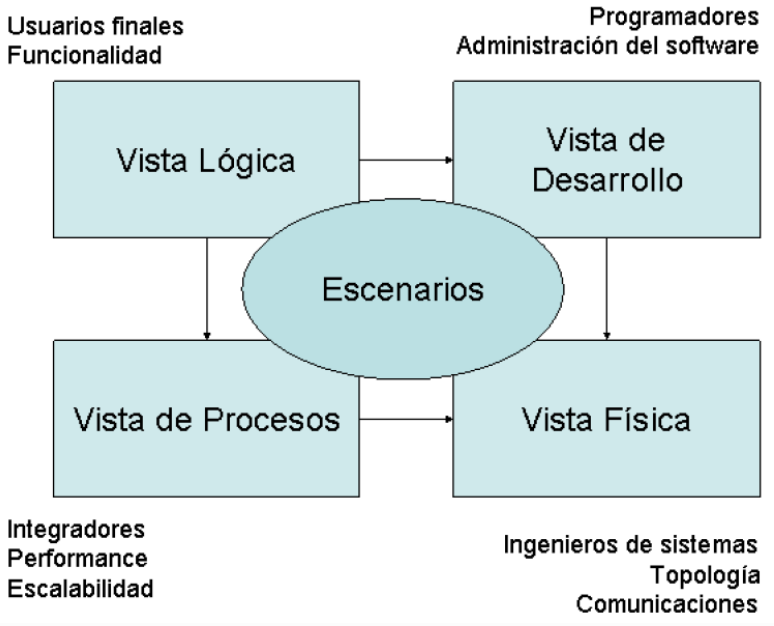


Figura 4. Modelo de vistas de arquitectura 4+1 (Kruchten, 1995, p. 2)

Vista física: Es en esta vista donde se determinan los componentes de índole física del sistema como pueden ser los nodos de interconexión, topología, etc. Toma en cuenta los requerimientos no funcionales del sistema como son la disponibilidad, tolerancia a fallos, rendimiento y escalabilidad. Los diversos elementos identificados en las vistas: lógica, de procesos y desarrollo, deben ser mapeados en diversos nodos de procesamiento.

+ 1 Vista de escenarios "Todas las vistas juntas": Se utiliza para unir las cuatro vistas descritas por medio de las secuencias de interacciones que se llegan a establecer entre sus elementos. Sus principales propósitos son: (conforme al Artículo publicado en IEEE Software 12(6), Noviembre 1995. Traducido por María Cecilia Bastarrica en Marzo 2006):

- "Funcionar como una guía para descubrir elementos arquitectónicos durante el diseño de arquitectura".

- "Como un rol de validación e ilustración después de completar el diseño de la arquitectura, en el papel y como punto de partida de las pruebas de un prototipo de la arquitectura".

A pesar de que el modelo no indica de manera explícita cómo relacionar los elementos de una vista con otra, estas mantienen una relación de correspondencia entre ellas, lo cual denota que las vistas no son del todo independientes. El fin principal de éste modelo de vistas es proveer un mecanismo para separar los diferentes aspectos del software ya que cada participante del proyecto tiene un interés diferente sobre el sistema: desarrollador, usuario final, administrador del proyecto, etc.

Para la notación de las vistas, comúnmente se utiliza UML puesto que es posible representar por medio de éste los diversos aspectos de la arquitectura, tal como se muestra en la siguiente tabla:

Vista	Descripción	Dirigida a	Considera	Diagrama UML
Lógica	Descomposición orientada a objetos.	Usuario final.	Los requerimientos funcionales: lo que el sistema debe proveer en términos de servicio a los usuarios.	De clases, estados, objeto, secuencia, comunicaciones.

Vista	Descripción	Dirigida a	Considera	Diagrama UML
Procesos	Descomposición de los procesos.	Integradores	Los requerimientos no funcionales: concurrencia, rendimiento, escalabilidad.	De actividad
Desarrollo	Descomposición de los subsistemas.	Programadores, administradores de software	La organización modular del software (jerarquía de capas, reusabilidad, restricciones de las herramientas de desarrollo)	De componentes, diagrama de paquetes.

Física	Mapeando el software hacia el hardware	Ingenieros de soporte, sistemas.	Requerimientos no funcionales relacionados con el hardware: topología, comunicaciones, etc.	De implementación.
Escenario	Poniendo todo junto	Todos los usuarios de las demás vistas y evaluadores.	Consistencia del sistema, validación. Es redundante con las otras vistas.	Casos de uso.

Tabla 5. Diagramas UML asociados a cada vista del modelo 4+1

1.2. La arquitectura en la fase de diseño de software

El proceso de desarrollo de software debe entenderse como un método ordenado donde se describe paso a paso lo que se hará y cómo se hará al momento de administrar un proyecto de desarrollo de software. Estos pasos que se mencionan deben estar incluidos a un proceso o metodología ampliamente aceptada, que de manera clara, se pueda ubicar y secuenciar lo necesario para poder llevar a cabo un proyecto de desarrollo de software. Algunas metodologías se clasifican por su extensión y ámbito. Hay metodologías ágiles, predictivas y descriptivas.

A continuación se menciona una metodología comúnmente usada en la industria del desarrollo del software, y que define de manera clara cómo se debe administrar un proyecto desde su concepción hasta su implementación: se trata de la metodología RUP.

1.2.1. La metodología RUP

La metodología RUP (del inglés *Rational Unified Process*), distingue varios pasos (fases) que se deben completar para poder decir que se realizó la administración del desarrollo de un proyecto de software:

- Venta

- Planeación
- Análisis
- Diseño
- Construcción
- Pruebas
- Implementación

Es importante aclarar que cada una de estas etapas son específicas de RUP, pero no difieren mucho de una teoría ampliamente aceptada e implementada por el proceso administrativo.

La parte donde se involucra directamente la AS es en la etapa del diseño de software, pues en esta llegan desde la fase anterior (análisis) los requerimientos que el cliente pide, y a partir de ellos es que se podrá comenzar a elaborar la mejor arquitectura a consideración del arquitecto.

Para tener de manera más clara la ubicación física de la AS dentro del modelo presentado por RUP, se muestra la siguiente imagen donde se señala la etapa de Diseño, junto con las etapas antecesoras y predecesoras:

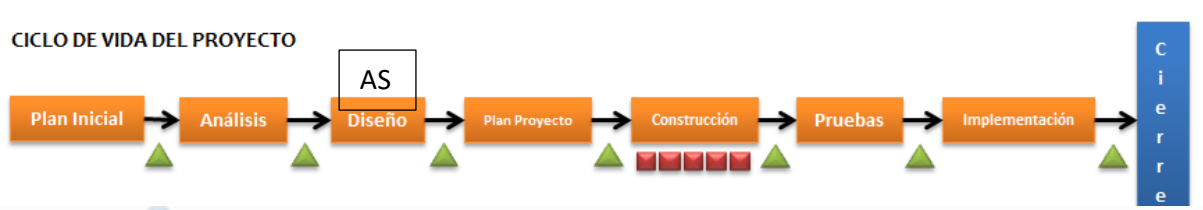


Figura 5. Ciclo de vida del proyecto de software modelo RUP

En la imagen anterior, la etiqueta AS denota la ubicación física del proceso de elaboración de la arquitectura del software, y se ve claramente el impacto que tendrá en la calidad del software completo, pues si no se hace de manera correcta, aunque el análisis haya sido aceptado por el cliente, al llegar a la etapa de construcción se llevarán una serie de errores (físicos, lógicos, de implementación, entre otros) y el resultado final estará destinado al fracaso.

No se debe confundir el diseño del software con la elaboración de la arquitectura, pues esta misma es una parte (subetapa) de la fase de diseño.

1.2.2. Propiedades y principios del diseño de software

La arquitectura de software se establece en la fase de diseño y puede conceptualizarse conforme a lo descrito por Roger Pressman (traducción del libro "Software Engineering – A practitioner's approach) "El diseño es prácticamente todo lo que un ingeniero de software desea realizar. Es el espacio donde la creatividad toma forma y donde los requerimientos, reglas del negocio y consideraciones técnicas convergen en la formulación de un producto o sistema. El diseño crea la representación o modelo del software proveyendo detalles acerca de su arquitectura necesaria para la implementación del sistema". Es en esta fase por lo tanto, donde se construye el plan para llevar adecuadamente el proyecto por el resto de las etapas hasta llegar a su adecuada implementación, prueba, mantenimiento y cierre. Esta fase toma los resultados de la fase de requerimientos y representa el "cómo" en el proceso de desarrollo del sistema.

El elemento de entrada para la fase de diseño lo constituyen los requerimientos, documentados y autorizados. Los objetivos principales de esta fase consisten en generar las especificaciones detalladas de diseño y establecer la base guía para el desarrollo del proyecto. Pero, ¿por qué es importante realizar el diseño del software? Si trasladamos el concepto por ejemplo a la construcción de un carro, televisión o incluso un puente, no es factible su construcción sin antes realizar el diseño adecuado, con los cálculos que involucra, planos, etc., y tampoco sería posible realizar modificaciones sin contar con documentación detallada. Lo mismo sucede con el software, no es posible que un desarrollador construya un determinado sistema sin tener un plan detallado.

Como se mencionó anteriormente, las actividades o pasos que conllevan a la obtención de un diseño de arquitectura de software forman parte de la fase de diseño de cualquier metodología

de desarrollo y existen diferentes principios y técnicas en la literatura de la Ingeniería de Software que permitirán obtener un buen resultado de diseño.

Es mediante el diseño de un sistema de software como se establecen las consideraciones técnicas y reglas de negocio generando una representación o modelo de software que brinda los detalles acerca de la arquitectura, estructuras de datos, interfaces y componentes necesarios para la implementación del sistema.

Algunas técnicas esenciales del diseño son las siguientes (Árias, 2014):

- División y conquista: Descomposición del problema.
- Abstracción: Representación de las características esenciales.
- Encapsulamiento: Ocultamiento de los detalles de implementación.
- Modularización: Descomposición del sistema en módulos.
- Acoplamiento y cohesión: Interdependencia entre módulos y relación de las tareas realizadas en un mismo módulo.
- Estructura jerárquica.
- Separación de políticas de la ejecución de algoritmos
- Separación de interfaces de sus implementaciones

Entre los principios que deben considerarse en el diseño de software se consideran indispensables, los siguientes (Roger S. Pressman, Software Engineering – A practitioner's approach, 2010):

- El diseño debe constituir una guía comprensible y legible para el equipo de desarrollo, de pruebas y posteriormente, de soporte.
- Debe proveer una imagen completa del software, que direccione adecuadamente datos, funcionalidad y relaciones entre los dominios desde la perspectiva de implementación.
- Considerar varios enfoques en base a los requerimientos definidos en la fase de análisis.
- El diseño debe estructurarse de tal manera que sea adaptable al cambio.

- Implementar un control de calidad durante el proceso de creación, no al final del diseño, lo cual reduce el tiempo de desarrollo.
- Revisiones formales periódicas para minimizar errores conceptuales.
- La consistencia del diseño y la uniformidad se tornan cruciales para sistemas de gran tamaño.
- "El diseño no es escribir código y escribir código no es diseñar" (Ingeniería del software, Tema 5: Principios del diseño del software, Universidad de Salamanca, p. 22)

1.2.3. Proceso de diseño de software

En el proceso de diseño de software, es muy complicado que un diseñador obtenga un producto final de forma inmediata, el desarrollo del diseño es iterativo a través de un número diferente de versiones y el punto inicial lo constituye un diseño informal que es redefinido agregando información para generar un resultado consistente y completo.

Las actividades que se realizan en éste proceso conforme a lo establecido por Ian Sommerville (2005) son las siguientes:

1.- Dividir requerimientos

Los requerimientos identificados y solicitados por el cliente, resultado de la fase de análisis, conforman el elemento de entrada de la fase de diseño, sin esta información no es posible empezar a diseñar. El fin de la primera actividad del diseño es dividir los requerimientos obtenidos en grupos afines que pueden estar dados por: áreas de funcionalidad, procesos (reglas de negocio, procedimientos de bases de datos, reportes, etc.) o cualquier otra subdivisión que el diseñador considere adecuada.

2.- Identificar subsistemas

Es muy parecida a la actividad anterior, ya que es muy común que los grupos de requerimientos resultantes estén relacionados con los subsistemas. Éstos subsistemas

deben identificarse de tal manera que cumplan cabalmente con los requerimientos ya sea de manera individual o colectiva. Cabe mencionar que esta actividad puede verse influenciada por factores organizacionales y del entorno. También como parte de esta actividad, se identifican el marco para el control y comunicación entre los subsistemas.

3.- Asignar requerimientos a los subsistemas

Si la división de requerimientos se utiliza para la identificación de los subsistemas esta actividad resulta muy sencilla de realizar, en la cual se deben describir las funciones que serán realizadas por cada subsistema. Sin embargo, se puede deducir que en la práctica no siempre resulta igual la división de requerimientos a la identificación de los subsistemas, incluso existen ocasiones que debe realizar un re-análisis de ciertos requerimientos para adaptarlos, dentro de un proceso iterativo, a los subsistemas identificados.

4.- Especificar la funcionalidad de los subsistemas

Una vez que se definieron los subsistemas que formarán parte de la aplicación, es necesario determinar su funcionalidad, así como especificar las relaciones y formas de interconexión de los mismos. Debe existir una especificación abstracta de los servicios que provee y las restricciones bajo las cuales se va a operar.

5.- Definir las interfaces de los subsistemas

En esta actividad se realiza el diseño requerido para la interface de cada uno de los subsistemas previamente definidos lo que permitirá desarrollarlos en paralelo. El diseño de las formas de interface de usuario es un componente importante en el proceso que incluso puede influenciar otras decisiones sobre el mismo diseño. Para el diseño de las interfaces se toman en cuenta aspectos como: visibilidad del estatus del sistema, consistencia, flexibilidad y eficiencia de uso, ayuda a los usuarios sobre el sistema, entre otros.

Entre las actividades descritas existe una alta retroalimentación e iteración ya que al momento de estar definiendo los resultados de cada etapa es muy común que surjan cuestionamientos que incluso provoquen que sea necesario rehacer el trabajo ya realizado. Es muy importante mencionar que en caso de que ya exista un sistema funcional éste limita considerablemente las elecciones del diseño. Una manera de representar estos procesos relacionados es mediante una espiral conforme a lo indicado en la imagen siguiente:

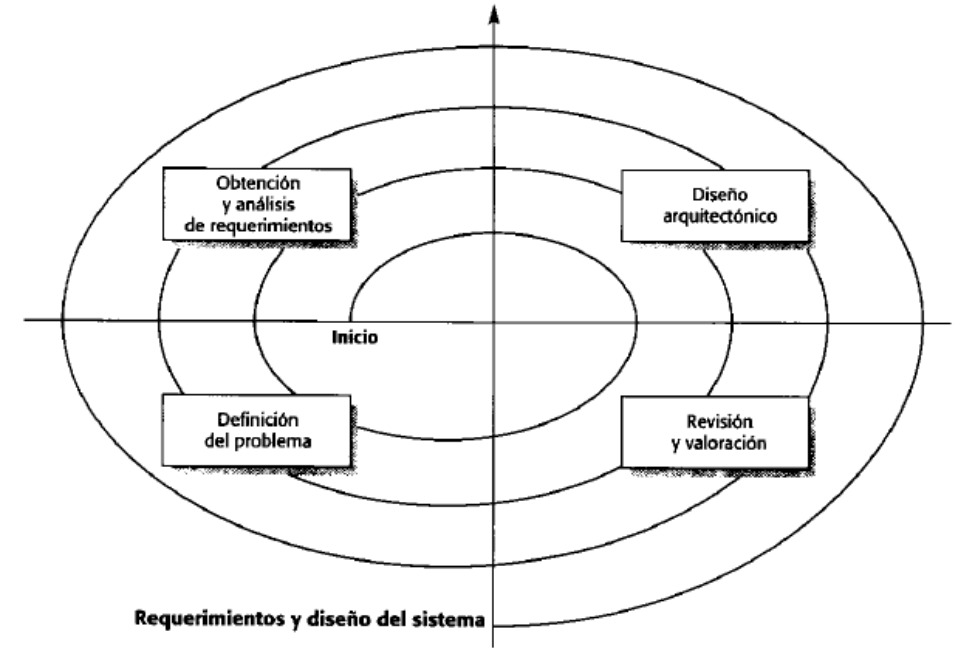


Figura 5. Modelo en espiral de requerimientos y diseño (Ian Sommerville – Ingeniería del Software)

Este diagrama expresa cómo los requerimientos afectan las decisiones de diseño y viceversa por lo que toma sentido entrelazar los procesos. El diseño comprende una solución que combina hardware, software y operaciones humanas y deberá proveer una arquitectura de software adecuada. El resultado final de la fase de diseño será entonces una guía para los programadores traducida en una arquitectura de software que asegure un perfecto entendimiento entre los colaboradores y facilite la implementación de la aplicación informática.

La transición de la fase de diseño de software hacia arquitecturas tradicionales o emergentes marca un hito crucial en el desarrollo tecnológico contemporáneo. Entre las arquitecturas tradicionales, destacan las monolíticas, que han sido piedras angulares en el diseño de sistemas durante décadas, al integrar todas las funcionalidades en un solo bloque de código, simplificando la gestión y el despliegue, aunque a menudo limitan la escalabilidad y la capacidad de adaptación a cambios. Asimismo, las arquitecturas cliente/servidor han sido fundamentales, distribuyendo las responsabilidades entre un cliente que solicita servicios y un servidor que responde a esas solicitudes.

Contrastando con estas arquitecturas, las arquitecturas emergentes, como las orientadas a microservicios, representan un enfoque contemporáneo al dividir la aplicación en servicios independientes, facilitando así la escalabilidad y la flexibilidad con actualizaciones y despliegues independientes.

Otra tendencia destacada son las arquitecturas de contenedores, que ofrecen un entorno liviano y portátil para ejecutar aplicaciones, simplificando la implementación coherente en diversos entornos, desde el desarrollo hasta la producción.

Las arquitecturas de nube representan un cambio fundamental en la forma en que diseñamos, implementamos y gestionamos aplicaciones, aprovechando los recursos en la nube para ofrecer escalabilidad automática, redundancia y accesibilidad global.

Por último, las arquitecturas de inteligencia artificial introducen la integración de capacidades de aprendizaje automático y procesamiento de datos masivos, influyendo significativamente en la arquitectura de software y permitiendo la construcción de sistemas adaptables y evolutivos.

En conjunto, estas arquitecturas representan una evolución en la concepción y diseño de software, ofreciendo soluciones flexibles, escalables y adaptativas para satisfacer las demandas de un entorno tecnológico en constante cambio.

Tanto las arquitecturas tradicionales como las emergentes, ofrecen perspectivas valiosas para la construcción de software. Esta evolución en la concepción y diseño del software no solo responde a las demandas de un entorno tecnológico en constante cambio, sino que también sienta las bases para la creación de soluciones flexibles, escalables y adaptativas. La comprensión del funcionamiento de elementos tradicionales y emergentes te posicionará como futuro profesional para enfrentar con éxito los futuros desafíos tecnológicos, garantizando la entrega de software de alta calidad en un mundo digital dinámico.

Cierre de la unidad

Has concluido la primera unidad del curso. A través de una revisión cronológica te has introducido al conocimiento de los principales lenguajes descriptores de arquitectura y su posible aplicación para el diseño de una arquitectura de software y también has estudiado las vistas de la arquitectura “4+1” ampliamente difundido. Como cierre, has adquirido conocimientos sobre algunas de las arquitecturas, tanto tradicionales como emergentes, que desempeñan un papel fundamental en el panorama actual de la ingeniería de software.

La comprensión total de los ejemplos y definiciones presentadas a lo largo de la unidad será de importancia para las unidades siguientes y para su correcta utilización en ambientes de producción real.

Es aconsejable que revises nuevamente la unidad en caso de que los temas que se acaban de mencionar no te sean familiares o no los recuerdes, de no ser este tu caso, ya estás preparado(a) para seguir con la unidad dos, en donde continuarás con la revisión de los patrones y modelos de arquitectura más utilizados. Todo ello con el fin de obtener el conocimiento necesario para comenzar a realizar propuestas de arquitectura al final de la unidad 2.

Es importante que identifiques un software de código libre y realices una descripción formal de arquitectura, basándote en un lenguaje de definición de arquitectura, instálalo en tu computadora personal para que realices pruebas de descripción y veas la aplicación de los conceptos presentados.

Fuentes de consulta

Bass, L., Clements, P. y Kazman, R. (2003). *Software architecture in practice* (2.ª ed.). Addison-Wesley.

Castro, L., Cervantes, H. y Velasco, P. (2015). *Arquitectura del software: Conceptos básicos*. Editores S.A. de C.V.

Durango, A. y Arias, Á. (2014). *Ingeniería y arquitectura de software*. CreateSpace.

Kruchten, P. (1995). Architectural blueprints: The "4+1" view model of software architecture. *IEEE Software*, 42–50. https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Pressman, R. (2010). *Software engineering: A practitioner's approach* (7.ª ed.). McGraw-Hill Higher Education.

Reynoso, C. y Kicillof, N. (2004). *Lenguajes de descripción de arquitectura*. Universidad de Buenos Aires.

Sommerville, I. (2005). *Ingeniería del software* (7.ª ed.). Pearson Education.