Universidad de las Ciencias Informáticas

Facultades 3 y 4

Título: **"Una guía práctica de Arquitectura de Software"**

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores:

Osvaldo Díaz Verdecia

Virgen Damaris Quevedo Campins

Tutor:
Ing. René Lazo Ochoa

Co-Tutor:
Ing. César Lage Codorniú

(Mayo), 2009
"Año 50 de la Revolución"

AGRADECIMIENTOS

De Osvaldo:

A mi mamá, porque siempre me exigió y estuvo pendiente de cada logro y fracaso mío, y porque sé que esta también fue su tesis.
A mi papá, porque me enseño todo lo que sé de la vida y nunca hubiera llegado a donde estoy sin sus consejos.
A mi hermano, que sé que mejorará mis pasos, que ha estado siempre conmigo y que siempre estará presente, apoyándome, en cada paso que dé en la vida.
A mi abuela, porque siempre me apoyo incluso cuando nadie más lo hizo.
A mi abuelo, que aunque ya no esta conmigo, le agradezco toda la confianza y orgullo que depositó en mí y sé que no lo defraudé.
A mi tío Framil y mis primos, que siempre estuvieron apoyándome en todo lo que necesite.
A mi novia, que juntos soñamos esta tesis y que me aguanto todos estos años.
A Jonny que fue mi amigo en las buenas y en las malas.
A mis tutores, que hicieron posible este trabajo.
A toda la gente del equipo de arquitectura, Javier, Ariecer, Adriana, Noel, David, Grettel, Yannier, que me ayudaron mucho desde que entre en ERP y con los que me divertí a lo grande.
A mis amigos, que estuvieron pendientes de cada cosa que necesité.
A Tania, una amiga de mi Mamá, que se debe saber la tesis de memoria de tantas revisiones que le hizo y que tanto saber nos aportó.
A todas aquellas personas anónimas, que no lo son tanto, que saben que me han ayudado en todo el transcurso de mi vida y que el único motivo que no estén aquí es mi pésima memoria.

De Damaris:

Este trabajo no fue solo resultado del esfuerzo de mi compañero de tesis y mío, sino de muchas personas que contribuyeron directa e indirectamente en la cumplimentación del mismo. Nuestro co-tutor por ejemplo que siempre nos ayudo muchísimo, incluso paso a ser co-tutor por toda la ayuda que nos presto que fue tanta como la del tutor, a este último también agradecerle por siempre darnos la confianza de que "ya eso esta terminado!!!" aunque nos faltara casi toda la tesis, a las tesistas Gretel, Adriana y Yaimi, que también dieron su aporte a dos capítulos de nuestra tesis. Al equipo de arquitectura del proyecto ERP (Orlando, Ariel, Erick, Borbón, Oiner, Dionisdel y César).
A mi suegra, y sus compañeras de trabajo que tantas revisiones nos hicieron y tantas faltas nos corrigieron, siendo de vital importancia para la calidad del trabajo, y que casi se tomaron más molestias con la tesis que nosotros.
A mi novio y compañero de tesis, por aguantarme como novio mis luchas con este trabajo y mi desesperación ante las complicaciones, y a su cooperación como compañero de tesis sin dejarme toda la carga a mi, ni presionarme, al contrario, siempre aliviándome trabajo, y apoyándome como nadie.
Y lo dejo para último porque para mí es el más especial: a mi padrastro agradecerle todo lo que me ayudó, y a mi mamá que sin entender un ápice de lo que le hablaba se acostaba a escucharme exponer la tesis hasta marearla. A todas esas personas, que son tan autoras de este trabajo como nosotros dos, mis más sinceros agradecimientos.

Dedicatoria

De Osvaldo:

A mi Papá, mi Mamá, mi Hermano, mi Abuela, mi Tío, mis Primos y especialmente a mi abuelo que aunque ya no está conmigo, siempre estará a mi lado.

De Damaris:

A mi mamá (Ana Cecilia) en primer lugar, a mi padrastro (Rolando) siempre tan bueno, y a mi papá (Raúl) que tanta ilusión le hace mi graduación. Pero muy en especial a mi abuelita Mercedes, una de las personas que más quiero en este mundo, y que por lo tanto, lo primero más importante que hago, debo dedicárselo a ella también.

Resumen

En este trabajo se realiza un Estado del Arte de la Arquitectura de Software, a nivel mundial, se presentan sus principales conceptos, las vistas que la integran según diversos expertos y una definición propia por parte de los autores, de lo que estas representan actualmente, así como una guía de actividades a desarrollar en cada una de estas vistas. También se trata sobre la importancia del Arquitecto de Software en la actualidad, sus responsabilidades y competencias según las diferentes temáticas en las que se vea envuelto; cómo estructurar un expediente de la Arquitectura de Software y las partes de que está compuesto, así como qué son las Pruebas de Concepto y su importancia.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Introducción

La disciplina del conocimiento de la Arquitectura de Software, está diseminada en el mundo con gran polarización en el consenso conceptual. Las literaturas se encuentran parcializadas en corrientes específicas y muy asentadas en alguna tecnología. Tiene un alto grado de integración con otras disciplinas del desarrollo de software y un impacto alto en los resultados de los productos. Además existe una elevada multidisciplinariedad en la temática y un alto grado de complejidad del conocimiento que agrupa. La Arquitectura de Software no se estudia en ninguno de los programas de carrera del país, la formación en la misma es producto del auto aprendizaje de los especialistas, de aquí que su implementación práctica en la Universidad de las Ciencias Informáticas (UCI) no logre cubrir, normar y dirigir todos los aspectos integrados del desarrollo de la misma. En este caso, la poca experiencia de la disciplina en la Universidad ha implicado que se incurra en costosos errores en el desarrollo y en muchos casos la afectación de los atributos de calidad del producto. Como consecuencia surge el **problema científico:** ¿Cómo organizar, construir y evaluar la Arquitectura de Software?

El **objeto de estudio** de esta investigación se centra en la Ingeniería de Software, siendo el **campo de acción** la Arquitectura de Software

Para resolver el problema planteado el **objetivo general** que se persigue con este trabajo es realizar una guía que permita la organización, construcción y evaluación de los aspectos arquitectónicos en un proyecto de software.

Los **objetivos específicos** de este trabajo son:

- Elaborar un marco teórico de la Arquitectura de Software.
- Caracterizar las disciplinas que se agrupan en la Arquitectura del Software.
- Identificar los diferentes roles que se definen dentro de dichas disciplinas, así como sus responsabilidades y competencias.
- Definir un expediente para organizar y almacenar la documentación generada en la Arquitectura de Software de un proyecto.
- Mencionar y explicar las pruebas de concepto necesarias para evaluar la Arquitectura de Software.

Este trabajo de Diploma está estructurado en 5 Capítulos que se desglosan a continuación y donde encontrará todo lo necesario y suficiente para comenzar a aplicar una arquitectura al desarrollar un software:

- Capítulo No. 1: se aborda el Estado del Arte y campo de acción de la Arquitectura de Software, exponiendo además su historia y principales definiciones.
- Capítulo No. 2: se especifican las diferentes disciplinas en las que se divide la Arquitectura de Software y se propone una guía de actividades a realizar en cada una de estas disciplinas y como se llevan a cabo.
- Capítulo No. 3: se tratan aspectos del Rol del Arquitecto como sus principales responsabilidades y competencias, brindando referencias de las principales compañías, centros de investigación y organizaciones que han estudiado el tema.
- Capítulo No. 4: se propone un expediente donde organizar todo la documentación generada en un proyecto por la Arquitectura de Software.
- Capítulo No. 5: se exponen las principales Pruebas de Concepto que se aplican hoy día en el mundo para evaluar la arquitectura y las técnicas utilizadas para ello.

Teniendo en cuenta lo anteriormente señalado se espera que llegue a ser una guía de utilidad para cualquier persona, profesor, estudiante, profesional, incluso los que incursionen por primera vez en la rama de la informática con total desconocimiento al respecto, a fin de que le resulte beneficiosa para dominar lo esencial del tema. Los autores esperan además que sea una herramienta fácil y cómoda, donde de manera rápida y concreta el lector se adentre en la Arquitectura de Software, temática importante en este mundo actual en que las sociedades están dirigidas cada vez más a la informatización de cada uno de sus procesos.

Capítulo 1. Estado del Arte de la Arquitectura de Software.

Introducción del Capítulo

En el presente capítulo se sientan las bases de la Arquitectura de Software, dónde surge, cómo fueron sus comienzos, así como se analizan las principales corrientes teóricas y se examina la teoría que la origina. También se exponen los principales conceptos que se manejan en la Arquitectura de Software como son los estilos, frameworks*[1], vistas y Lenguajes de Descripción Arquitectónica (ADLs)*. Se hace un recuento de los procesos y metodologías más usados en la actualidad y por qué, y se dan a conocer los diferentes tipos de escenarios, lo que significan y lo que en estos días representan.

1.1 Historia de la Arquitectura de Software

Con el surgimiento de la programación se da comienzo a un nuevo universo: la producción de software. A medida que los mismos fueron ganando en complejidad, dada la necesidad de resolver problemas más complicados y abarcadores se identificaron propiedades comunes acompañadas de soluciones muy similares en su estructura. Fue en el año 1968, en el que Edsger Dijkstra de la Universidad Tecnológica de Holanda, propuso que se hiciera una estructuración correcta de los sistemas de software antes de lanzarse a programar (Dijkstra). Dijkstra, quien sostenía que las Ciencias de la Computación eran una rama aplicada de las Matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes y que se superponen “como capas de cebolla”. Inventó o ayudó a precisar además docenas de conceptos: el algoritmo del camino más corto, los stacks, los vectores, los semáforos, los abrazos mortales. De sus ensayos arranca la tradición de hacer referencia a “niveles de abstracción” que ha sido tan común en la arquitectura subsiguiente. Aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego expresarían Niklaus Wirth[2] como el perfeccionamiento gradual y DeRemer y Kron como

[1] El * significa que la palabra está referenciada en el Glosario de Términos donde se recogen todos los términos que puedan generar dudas o que no sean muy conocidos, y además incluye los acrónimos.

[2] Su artículo de desarrollo de un programa por refinamiento sucesivo ("program development by stepwise refinement") es considerado un texto clásico en la ingeniería del software, así como su libro *Algoritmos + Estructuras de datos = Programas*, recibió un amplio reconocimiento, y aún hoy en día es útil en la enseñanza de la programación. Recibió el Premio Turing por el desarrollo de estos lenguajes de programación en 1984. Se jubiló en 1999.

programación en grande, ideas que poco a poco irían decantando primero los ingenieros y después los arquitectos.

La Ingeniería de Software se fundó en 1968, cuando F.L. Bauer usó ese sintagma por primera vez en la conferencia de la OTAN de Garmisch, Alemania. Un año más tarde en la conferencia de la NATO[3] de 1969, Sharp formuló sorprendentes apreciaciones comentando las ideas de Dijkstra alegando que se debía hacer algo aparte de la Ingeniería de Software refiriéndose así a la Arquitectura de Software (Arquitectura de Software) diferenciándola de la Ingeniería de Software.

En 1969 Fred Brooks Jr y Ken Iverson llamaban arquitectura a la estructura conceptual de un sistema en la perspectiva del programador. En 1971, C. R. Spooner tituló uno de sus ensayos “Una Arquitectura de Software para los 70s”, sin que la mayor parte de la historiografía de dicha disciplina registrara ese antecedente.

En 1975, Brooks, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa, empleando una nomenclatura que ya nadie aplica de ese modo. En el mismo texto, identificaba y razonaba sobre las estructuras de alto nivel y reconocía la importancia de las decisiones tomadas a ese nivel de diseño. También distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo. Aunque el concepto de Arquitectura de Software actual y el de Brooks difieren en no escasa medida, el texto del libro The mythical man-month sigue siendo, un cuarto de siglo más tarde, consultado en temas de Ingeniería de Software.

En la misma época de los 70, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada. Parnas desarrolló temas tales como módulos con ocultamiento de información, estructuras de software y familias de programas, enfatizando siempre la búsqueda de calidad del software, medible en términos de economías en los procesos de desarrollo y mantenimiento.

El primer estudio en que aparece la expresión “Arquitectura de Software” en el sentido en que hoy lo conocemos es sin duda el de Dewayne Perry y Alexander Wolf en 1992, aunque el trabajo se fue gestando desde 1989. En él, los autores proponen concebir la Arquitectura de Software por analogía con la arquitectura de edificios. El artículo comienza diciendo exactamente:

3 NATO: North Atlantic Treaty Organization

"*El propósito de este papel es construir el fundamento para la Arquitectura de Software. Primero desarrollaremos una intuición para la Arquitectura de Software recurriendo a diversas disciplinas arquitectónicas bien definidas. Sobre la base de esa intuición, presentamos un modelo para la Arquitectura de Software que consiste en tres componentes: elementos, forma y razón (rational). Los elementos son elementos ya sea de procesamiento, datos o conexión. La forma se define en términos de las propiedades de, y las relaciones entre, los elementos, es decir, restricciones operadas sobre ellos. La razón proporciona una base subyacente para la arquitectura en términos de las restricciones del sistema, que lo más frecuente es que se deriven de los requerimientos del sistema. Discutimos los componentes del modelo en el contexto tanto de la arquitectura como de los estilos arquitectónicos...."* (1)

Estos autores prosiguen reseñando el progreso de las técnicas de diseño de los 70, y ya en la década de 1980 se perfeccionaron las técnicas descriptivas y las notaciones formales, permitiéndose razonar mejor sobre los sistemas de software. Para la caracterización de lo que sucederá en la década siguiente ellos formulan esta otra frase que ha quedado inscripta en la historia mayor de la especialidad:

"La década de 1990, creemos, será la década de la Arquitectura de Software. Usamos el término "arquitectura" en contraste con "diseño", para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de los arquitectos de software) y de estilo. Es tiempo de re-examinar el papel de la Arquitectura de Software en el contexto más amplio del proceso de software y de su administración, así como señalar las nuevas técnicas que han sido adoptadas." (1)

Dando cumplimiento a las profecías de Perry y Wolf, la década de 1990 fue sin duda la de la consolidación y diseminación de la Arquitectura de Software en una escala sin precedentes. Las contribuciones más importantes surgieron en torno al Instituto de Ingeniería de la Información de la Universidad Carnegie Mellon.

En la misma década, surge también la programación basada en componentes, que en su momento de mayor impacto impulsó a algunos arquitectos mayores, como Paul Clements, al afirmar que la Arquitectura de Software promovía un modelo que debía ser más de integración de componentes pre-programados que de programación.

Un segundo gran tema de la época fue el surgimiento de los patrones, el creador de esta idea fue Christopher Alexander, quien incidentalmente fue arquitecto de edificios; Alexander desarrolló en diversos estudios de la década de 1970 temas de análisis del sentido de los planos, las formas, la edificación y la construcción, en prueba de un modelo constructivo y humano de arquitectura, elaborada de forma que tenga en cuenta las necesidades de los habitantes.

La Arquitectura de Software de este período realizó su trabajo de homogenización de la terminología, desarrolló la tipificación de los estilos arquitectónicos y elaboró ADLs. También se consolidó la concepción de las vistas arquitectónicas, reconocidas en todos y cada uno de los frameworks generalizadores que se han propuesto como son: 4+1, TOGAF*, RM/ODP, IEEE.

En el siglo XXI, la arquitectura aparece dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos. La producción de estas nuevas metodologías ha sido masiva, y una vez más tiene como epicentro el trabajo del Software Engineering Institute (SEI)[4] en Carnegie Mellon. La aparición de las metodologías basadas en arquitectura, junto con la popularización de los métodos ágiles en general y programación extrema en particular, han causado un reordenamiento del campo de los métodos, hasta entonces dominados por las estrategias de diseño "de peso pesado". Después de la Arquitectura de Software y de las tácticas radicales, las metodologías nunca volverán a ser las mismas. (1)

1.2 Principales corrientes teóricas de la arquitectura

En la actualidad no se ha publicado un estudio reconocido y sistemático acerca de las corrientes teóricas de la Arquitectura de Software, ni nada que analice las particularidades de cada una de ellas, aunque se pueden distinguir a grandes rasgos seis corrientes:

1. Arquitectura como etapa de ingeniería y diseño orientada a objetos.
2. Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.
3. Estructuralismo arquitectónico radical.
4. Arquitectura basada en patrones.
5. Arquitectura procesual.
6. Arquitectura basada en escenarios.

Una de las corrientes más recientes es la que promueve el SEI, quien se ha convertido en punto de referencia en la informática a nivel mundial.

Arquitectura como etapa de la Ingeniería de Software orientada a objetos

[4] El SEI es el Software Engineering Institute un instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos, en Pittsburgh, en 1984.

Esta corriente arquitectónica está basada en el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman, combinado estrechamente al mundo de UML* (Lenguaje Unificado de Modelado) debe decirlo aquí que es cuando aparece por primera vez en el trabajo) y Rational. En esta postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo. Las definiciones revelan que la Arquitectura de Software, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descriptas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten.

Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas

Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código.

Estructuralismo arquitectónico radical

Se trata de un desprendimiento de la corriente anterior, mayoritariamente europea, que asume una actitud más confrontativa con el mundo UML. En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la Arquitectura de Software y otra que procura definir nuevos metamodelos y estereotipos de UML como correctivos de la situación.

Arquitectura basada en patrones

Esta corriente se basa principalmente en la redefinición de los estilos como patrones POSA[5], el diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura.

Arquitectura procesual

Desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos, los arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci, Charles Weinstock, intentan establecer modelos de ciclo de vida y técnicas de diseño de requerimientos, diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la Arquitectura de Software. Otras variantes dentro de la corriente procesual se caracterizan por cambios en la etapa del proceso de extracción de la arquitectura, generalización y reutilización.

Arquitectura Basada en Escenarios.

La Arquitectura Basada en Escenarios es la corriente más nueva. Se trata de un movimiento predominantemente europeo con centro en Holanda. Recupera el nexo de la Arquitectura de Software con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. En esta corriente suelen utilizarse diagramas de casos de uso UML como herramientas ocasionales, dado que los casos de uso son uno de los escenarios posibles y que no están orientados a objeto. Los autores vinculados con esta modalidad han sido, aparte de los codificadores de ATAM, CBAM, QASAR y demás métodos del SEI, los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research: Mugurel Ionita, Dieter Hammer, Henk Obbink, Hans de Bruin, Hans Van Vliet, Eelke Folmer, Jilles Van Gurp y Jan Bosch. La presencia de holandeses es significativa; la Universidad de Eindhoven es, incidentalmente, el lugar en el que surgió lo que P. I. Sharp proponía llamar la escuela arquitectónica de Dijkstra.

[5] Pattern Oriented Software Architecture

Software Engineering Institute (SEI)

El Software Engineering Institute es un instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos, en Pittsburgh, en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense, la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon.

Es un referente en Ingeniería de Software por realizar el desarrollo del modelo SW-CMM* (1991) que ha sido el punto de arranque de todos los que han ido formando parte del modelo que ha desarrollado sobre el concepto de capacidad y madurez, hasta el actual CMMI*. Ha servido como un recurso nacional en la Ingeniería de Software, seguridad informática y la mejora del proceso. Como parte de la Universidad Carnegie Mellon, que es bien conocida por sus programas altamente valorados en ciencias de la computación y la ingeniería, el SEI opera en la vanguardia de la innovación técnica. Trabaja en estrecha colaboración con la defensa y las organizaciones gubernamentales, la industria y la academia para mejorar continuamente los sistemas de software.

Dentro del campo de la arquitectura el SEI juega un gran papel ya que provee técnicas y métodos para crear, documentar, evaluar y reconstruir una arquitectura robusta.

Aportan la herramienta Architecture Expert (ArchE) que es una herramienta de diseño auxiliar que ayuda a los arquitectos a explorar diseños arquitectónicos impulsados por los atributos de calidad. Arche muestra al arquitecto propuestas para mejorar la arquitectura actual y permite al arquitecto decidir cual es la mejor alternativa.

Métodos de SEI para el diseño de una Arquitectura de Software

Attribute-Driven Design (ADD) method: es un enfoque para definir una Arquitectura de Software en el que el proceso de diseño se basa en los atributos de calidad de software necesarios. ADD sigue un proceso recursivo que se descompone en un sistema o elemento del sistema, mediante la aplicación de tácticas de arquitectura y patrones que cumplan su atributo de calidad de conducción de requisitos.

Quality Attribute Workshop (QAW): proporciona un método para la identificación de atributos críticos de calidad en una arquitectura del sistema, tales como disponibilidad, rendimiento, seguridad, interoperabilidad y modificabilidad, que se derivan de la misión o metas del negocio.

Métodos del SEI para evaluar sistemas y arquitecturas de sistemas

Architecture Tradeoff Analysis Method (ATAM): utilizado para revelar cómo la arquitectura satisface los atributos de calidad y de riesgos, las sensibilidades, y compensaciones involucradas en el cumplimiento de los requisitos.

Cost-Benefit Analysis Method (CBAM): este método constituye guías de los ingenieros de sistemas y otros stakeholders para la utilidad económica y ventajas asociadas a las decisiones arquitectónicas.

Active Reviews for Intermediate Designs (ARID): utilizado para evaluar diseños tempranos o porciones del diseño para su viabilidad en satisfacción con los stakeholders que les concierne.

Método del SEI para la documentación de la arquitectura

Views and Beyond* (V&B) Approach: sostiene que la documentación de una Arquitectura de Software es una cuestión de elegir un conjunto de vistas de la arquitectura, la documentación de cada una de las vistas y la información que se aplica a más de una vista.

V & B incluye un método para la elección de las vistas relevantes sobre la base de las estructuras que le son inherentes en la Arquitectura de Software y sobre las necesidades y preocupaciones de la documentación de la arquitectura de los stakeholders. Se muestra cómo documentar las vistas y el modo de documentar la información que se aplica a través de las vistas. Cubre también información práctica, como la forma de combinar juiciosamente puntos de vista a fin de evitar la sobrecarga de documentación, y la forma de documentar la arquitectura utilizando UML.

1.3 La Arquitectura de Software.

A medida que crece la complejidad de las aplicaciones, y que se extiende el uso de sistemas distribuidos* y sistemas basados en componentes, los aspectos arquitectónicos del desarrollo de software están recibiendo un interés cada vez mayor, tanto desde la comunidad científica como desde la propia industria del software. El término arquitectura aparece cada vez con mayor frecuencia en la literatura sobre Ingeniería del Software. Para demostrar esto basta señalar que el Proceso Unificado (*Unified Process*) —recientemente presentado como el método de desarrollo de software asociado al Lenguaje Unificado de Modelado (UML), y que como este, pretende convertirse en un estándar *de facto*— se define como *centrado en la arquitectura*. (2)

Sin pretender establecer una definición completa ni definitiva, se considera como *Arquitectura* la estructura de alto nivel de un sistema de software, lo que incluye sus componentes, las propiedades

observables de dichos componentes y las relaciones que se establecen entre ellos. (3) Esta definición se centra en aspectos puramente descriptivos, y determina que cualquier sistema de software, o al menos cualquiera que tenga una cierta complejidad, tiene una arquitectura, independientemente de si esta arquitectura está representada en algún lugar de forma explícita, o incluso de si quienes desarrollaron el sistema eran conscientes de ella. Otras definiciones incluyen también aspectos de proceso y, de este modo, Garlan y Perry añaden a la arquitectura de un sistema de software los principios y reglas que gobiernan su diseño y su evolución en el tiempo. (4)

Teniendo en cuenta todos estos aspectos, e incorporando algunos otros, en la ya citada notación UML se define la arquitectura como el conjunto de decisiones significativas acerca de la organización de un sistema de software; la selección de los elementos estructurales a partir de los cuáles se compondrá el sistema y sus interfaces, junto con la descripción del comportamiento de dichas interfaces en las colaboraciones que se producen entre los elementos del sistema; la composición de esos elementos estructurales y de comportamiento para formar subsistemas de tamaño cada vez mayor; y el estilo o patrón arquitectónico que guía esta organización: los elementos y sus interfaces, las colaboraciones y su composición. (2)

Aparejado a este concepto de arquitectura, surge la Arquitectura de Software, como la disciplina, inscrita dentro de la Ingeniería del Software, que se ocupa del estudio de la arquitectura de los sistemas de software, y también como una de las tareas del proceso de desarrollo, enmarcada dentro de las actividades propias del diseño.

La Arquitectura de Software (Arquitectura de Software) es una disciplina naciente, que está dando sus primeros pasos. Las definiciones clásicas de la arquitectura la definen como:
Mary Shaw y David Garlan, sugieren que la Arquitectura de Software sea un nivel del diseño referido a las ediciones "…más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño". (5)
Esta definición es la base de la corriente estructuralista de la arquitectura representada por Mary Shaw y David Garlan de la escuela de Carnegie Mellon, esta tiene variantes con modelos de datos Medvidovic[6], radicales y formales Moriconi - SRI[7].

[6] Nenad Medvidovic: Profesor Asistente en el Departamento de Ciencias de la Computación en la Universidad del Sur de California y miembro del profesorado de la USC Centro de Ingeniería de Software (CSE).

Esta definición no es capaz de definir el momento en el ciclo de desarrollo del software donde se debe llevar a cabo la arquitectura, según esta, el desarrollo de la misma comienza en un momento entre la definición de los requisitos y la modelación del sistema o entre la modelación del sistema, el análisis y el diseño.

Rational Unified Process, 1999, propone que:

"Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuáles el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que dirige esta organización, los elementos y sus interfaces, sus colaboraciones y su composición" (2)

Una definición reconocida es la de Clements: "La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones."

En una definición semejante, hay que aclararlo, la idea de "componente" no es la de la correspondiente tecnología de desarrollo (COM*, CORBA*, Component Model*, EJB*), sino la de elemento propio de un estilo. Un componente es una entidad, a la que los arquitectos prefieren llamar "componente" antes que "objeto". (6)

Debido a la cantidad de definiciones existentes en el campo de la Arquitectura de Software, existe en general un acuerdo que se refiere a la estructura del sistema a grandes rasgos, estructura consistente en componentes y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño, pues la Arquitectura de Software es después de todo, una forma de diseño de software que se manifiesta tempranamente en el proceso de creación de un sistema; pero este diseño ocurre a un nivel más abstracto que el de los algoritmos y las estructuras de datos. Mary Shaw y David Garlan sugieren que dichas cuestiones estructurales incluyan organización a grandes rasgos y estructura global de control; protocolos para la comunicación, la sincronización y el acceso a datos; la asignación de funcionalidad a elementos del diseño; la distribución física; la composición de los elementos de diseño; escalabilidad y rendimiento; y selección entre alternativas de diseño.

---

[7] Mark Moriconi "Interactive Design and Verification: A Message Switching Network Example. The Use of Formal Specification of Software 1979", "A Designer/Verifiers's Assistant", PegaSys and the Role of Logic in programming Environments. Advanced Programming Environments 1986"

En una definición tal vez demasiado amplia, David Garlan establece que la Arquitectura de Software constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño. La definición “oficial” de Arquitectura de Software se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, que expresa lo siguiente:
“La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”. (4)

Ante la problemática del número y variedad de definiciones existentes de Arquitectura de Software, Mary Shaw y David Garlan proporcionaron una sistematización iluminadora, explicando las diferencias entre definiciones en función de distintas clases de modelos. Destilando las definiciones y los puntos de vista implícitos o explícitos, los autores clasifican los modelos de esta forma:

1) **Modelos estructurales:** Sostienen que la Arquitectura de Software está compuesta por componentes, conexiones entre ellos y usualmente otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizada por el desarrollo de lenguajes de descripción arquitectónica (ADLs).

2) **Modelos de framework:** Son similares a la vista estructural, pero su énfasis primario radica en la estructura coherente del sistema completo, en vez de concentrarse en su composición. Los modelos de framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software específicas de dominios, como CORBA, o modelos basados en CORBA, o repositorios de componentes específicos, como PRISM.

3) **Modelos dinámicos:** Enfatizan la cualidad conductual de los sistemas. “Dinámico” puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.

4) **Modelos de proceso:** Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento de proceso. Esta vista se ejemplifica con el actual trabajo sobre programación de procesos para derivar arquitecturas.

5) **Modelos funcionales:** Una minoría considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un framework particular.

Ninguna de estas vistas excluye a las otras, ni representa un conflicto fundamental sobre lo que es o debe ser la Arquitectura de Software. Por el contrario, representan un espectro en la comunidad de investigación sobre distintos énfasis que pueden aplicarse a la arquitectura: sobre sus partes constituyentes, su totalidad, la forma en que se comporta una vez construida, o el proceso de su construcción.

1.4 Conceptos fundamentales de la arquitectura. La teoría punto de partida.

La Arquitectura de Software se articula alrededor de algunos conceptos, principios esenciales y herramientas características, los cuáles se tratan a continuación.

Estilos

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales.

Sucintamente descritos, los estilos conjugan componentes, conectores*, configuraciones y restricciones. Al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo, incidentalmente, se sitúa en un orden de discurso y de método que el modelado orientado a objetos en general y UML en particular no cubren satisfactoriamente. La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación. A diferencia de los patrones de diseños, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos. Algunos estilos típicos son las arquitecturas basadas en flujo de datos, las peer-to-peer*[8], las de invocación implícita, las jerárquicas, las centradas en datos o las de intérprete-máquina virtual.

Lenguajes de descripción arquitectónica.

[8] Persona-a-Persona

Los lenguajes de descripción de arquitecturas, o ADLs, ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad, más o menos en contemporaneidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML. Los ADL difieren sustancialmente de UML, que al menos en su versión 1.x se estima inadecuado en su capacidad para expresar conectores en particular y en su modelo semántico en general para las clases de descripción y análisis que se requieren. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. Muy pocos arquitectos de industria parecen conocer los ADLs y son menos aun quiénes los utilizan como instrumento en el diseño arquitectónico de sus proyectos. Hay unos veinte ADLs de primera magnitud y tal vez unos cuarenta o cincuenta propuestos en ponencias que no han resistido el paso del tiempo o que no han encontrado su camino en el mercado.

Frameworks y Vistas

Existen unos cuantos organismos de estándares (ISO*, CEN*, IEEE*, OMG*) que han codificado diferentes aspectos de la Arquitectura de Software, aun que no en su totalidad, con el objetivo de homogeneizar la terminología, los modelos y los procedimientos. Los emergentes del trabajo de sus comités son especificaciones y recomendaciones de variada naturaleza, como RM-ODP*, RUP*, RDS*, MDA* o IEEE 1471-2000. Casi cualquier combinación de tres o cuatro letras del alfabeto corresponde al acrónimo de algún estándar a tener en cuenta. La Arquitectura de Software hace mención eventual de esas doctrinas y los académicos ocupan sillas preferenciales en los organismos, pero su tratamiento exhaustivo en la literatura de investigación decididamente no califica como uno de los grandes temas prioritarios. Las recomendaciones de los marcos se tratan con sumo respeto pero también con alguna reticencia, tal vez porque se estima que los organismos privilegiarían más un acuerdo regido por una necesidad de equidistancia que una adecuada fundamentación formal, porque cualquier versión del estándar que se mencione en un papel habrá caducado cuando se lo publique.

Hay una excepción, sin embargo. Tanto los marcos arquitectónicos como las metodologías de modelado de los organismos acostumbran ordenar las diferentes perspectivas de una arquitectura en términos de vistas (views). La mayoría de los frameworks y estrategias reconoce entre tres y seis vistas, que son las que se incluyen en el cuadro. Una vista es, para definirla sucintamente,… “un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado”. (7)

Tabla 1. Vistas en los marcos de referencia

Zachman (Niveles)	TOGAF (Arquitecturas)	4+1 (Vistas)	[(Grady Booch, 1999)] (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					

El marco de referencia para la arquitectura empresarial de John Zachman identifica 36 vistas en la arquitectura "celdas" basadas en seis niveles: scope, empresa, sistema lógico, tecnología, representación detallada y funcionamiento empresarial y seis aspectos: datos, función, red, gente, tiempo, motivación. En el uso frecuente de Arquitectura de Software se ha estimado que este modelo es excesivamente rígido y sobre-articulado. Parecería existir cierto consenso sobre su excesiva ambición y su posible obsolescencia. Algunos manuales de Ingeniería de Software por ejemplo *Ingeniería de Software: Teoría y Práctica* de **Shari Lawrence Pfleeger.** y el de **Roger Pressman.** *Ingeniería del Software: Un enfoque práctico*, suelen omitir toda referencia a este marco, que se estima perteneciente a una esfera de dirección de la información y estrategia empresarial, antes que inscrito en el campo de la arquitectura. El framework ha estado también bajo discusiones y críticas frecuentes. No obstante, hay que reconocer que tres de las vistas propuestas por Zachman tan tempranamente como en 1982 (conceptual, lógica y física) se corresponden con el modelo de vistas de los marcos de referencia posteriores.

El Modelo de Referencia para Procesamiento Distribuido Abierto (RM-ODP) es un estándar de ISO y de ITU (ex-CCITT) que define un marco para la especificación arquitectónica de grandes sistemas distribuidos. Define, entre otras cosas, cinco puntos de vista (viewpoints) para un sistema y su entorno: empresa, información, computación, ingeniería y tecnología. Los cinco puntos de vista no

corresponden a etapas de proceso de desarrollo o refinamiento. De los cuatro estándares básicos que componen el modelo, los dos primeros se refieren a la motivación general del mismo y a sus fundamentos conceptuales y analíticos; el tercero (ISO/IEC 10746-3; UTI-T X.903) a la arquitectura, definiendo los puntos de vistas referidos; y el cuarto (ISO/IEC 10746-4; UTI-T X.904) a la formalización de la semántica arquitectónica. RM-ODP se supone neutral en relación con la metodología, las formas de modelado y la tecnología a implementarse, pero recomienda el uso de lenguajes formales de especificación como LOTOS, ESTELLE, SDL o Z. Se supone que un modelo pensado con similares propósitos, como CORBA, debería ser 100% conforme con la referencia, pero en verdad no es así; CORBA, por ejemplo, no proporciona soporte para el viewpoint empresarial, su modelo computacional revela desviaciones significativas del ISO/IEC o el UTI-T correspondiente, su modelo de binding* es más restringido (carece de multicast* o plug and play*) y aunque hay RFPs* que documentan estas divergencias, hasta donde puede saberse permanecen aún sin resolver. En los viewpoints de ingeniería y tecnología las discrepancias son menores, pero son muchísimas, y en ninguno hay concordancia en la nomenclatura. Los modelos mayores de industria como COM o J2EE son todavía más divergentes y las verificaciones de conformidad son un trámite extremadamente complejo. RM-ODP, además, no especifica nada acerca de conectores entre sistemas, integración de tecnologías “Legacy”* o soporte de sistemas multi-paradigmas. Hoy en día la interoperabilidad se garantiza más a través de tecnologías comunes de formato y protocolo ligadas a XML*, SOAP*, BPEL4WS* o WS-I* (o mediante MDA, o programando un wrapper*) que por conformidad con esta clase de recomendaciones en todos los puntos de un proceso distribuido.

C4ISR (Sistemas de Comando, Control, Comunicaciones, Computación, Inteligencia, Vigilancia y Reconocimiento) Architecture Framework es el marco de referencia arquitectónico promovido por el Departamento de Defensa de Estados Unidos (DoD). Algunos de los otros marcos listados en esta sección se inspiran en él, como es el caso de TOGAF. En la versión 2 de C4I, completada en diciembre de 1997, la definición de arquitectura reconocida es exactamente la misma que después se promulgaría como canónica en IEEE* 1471. Las vistas arquitectónicas homologadas son la Operacional (que identifica relaciones y necesidades de información), la de Sistemas (que vincula capacidades y características a requerimientos operacionales) y la Técnica (que prescribe estándares y convenciones).

El marco de referencia arquitectónico de The Open Group (TOGAF) reconoce cuatro componentes principales, uno de los cuáles es un framework de alto nivel que a su vez define cuatro vistas: Arquitectura de Negocios, Arquitectura de Datos/Información, Arquitectura de Aplicación y Arquitectura Tecnológica. The Open Group propone un modelo de descripción arquitectónica, Architecture Description Method (ADM) que se supone independiente de las técnicas de modelado, aunque en la versión 7 se propone Metis* como herramienta.

En 1995 Philippe Kruchten propuso su célebre modelo “4+1”, vinculado al Rational Unified Process (RUP), que define cuatro vistas diferentes de la Arquitectura de Software: 1. La vista lógica, que comprende las abstracciones fundamentales del sistema a partir del dominio de problemas. 2 La vista de proceso: el conjunto de procesos de ejecución independiente a partir de las abstracciones anteriores. (3) La vista física: un mapeado del software sobre el hardware. 4 La vista de desarrollo: la organización estática de módulos en el entorno de desarrollo. El quinto elemento considera todos los anteriores en el contexto de casos de uso. Lo que académicamente se define como Arquitectura de Software concierne a las dos primeras vistas. El modelo 4+1 se percibe hoy como un intento de reformular una arquitectura estructural y descriptiva en términos de objeto y de UML. Con todo, las cuatro vistas de Kruchten forman parte del repertorio estándar de los practicantes de la disciplina.

En su introducción a UML 1.3, Grady Booch, James Rumbaugh e Ivar Jacobson han formulado un esquema de cinco vistas interrelacionadas que conforman la Arquitectura de Software, caracterizada en términos parecidos a los que uno esperaría encontrar en el discurso de la vertiente estructuralista. En esta perspectiva, la Arquitectura de Software es un conjunto de decisiones significativas sobre:

1. La organización de un sistema de software.
2. La selección de elementos estructurales y sus interfaces a través de los cuáles se constituye el sistema.
3. Su comportamiento, según resulta de las colaboraciones entre esos elementos.
4. La composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente mayores.
5. El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos y sus interfaces, sus colaboraciones y su composición.

Los autores proporcionan luego un esquema de cinco vistas posibles de la arquitectura de un sistema:

1. La vista de casos de uso, como la perciben los usuarios, analistas y encargados de las pruebas;
2. La vista de diseño que comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución;
3. La vista de procesos que conforman los hilos y procesos que forman los mecanismos de sincronización y concurrencia;
4. La vista de implementación que incluye los componentes y archivos sobre el sistema físico;
5. La vista de despliegue que comprende los nodos que forma la topología de hardware sobre la que se ejecuta el sistema. Aunque las vistas no están expresadas en los mismos términos estructuralistas que sobresalen en su caracterización de la arquitectura, y aunque la relación

entre vistas y decisiones arquitectónicas es de simple yuxtaposición informal de ideas antes que de integración rigurosa, es natural inferir que las vistas que más claramente se vinculan con la semántica arquitectónica son la de diseño y la de proceso.

En los albores de la moderna práctica de los patrones, Buschmann y otros presentan listas discrepantes de vistas en su texto popularmente conocido como POSA. En la primera se las llama “arquitecturas”, y son:

1. Arquitectura conceptual: componentes, conectores.
2. Arquitectura de módulos: subsistemas, módulos, exportaciones, importaciones.
3. Arquitectura de código: archivos, directorios, bibliotecas, inclusiones.
4. Arquitectura de ejecución: tareas, hilos, procesos.

La segunda lista de vistas, por su parte, incluye:

1. Vista lógica: el modelo de objetos del diseño, o un modelo correspondiente tal como un diagrama de relación.
2. Vista de proceso: aspectos de concurrencia y sincronización.
3. Vista física: el mapeo del software en el hardware y sus aspectos distribuidos.
4. Vista de desarrollo: la organización estática del software en su entorno de desarrollo.

Esta segunda lista coincide con el modelo 4+1 de Kruchten, pero sin tanto énfasis en el quinto elemento.

Bass, Clements y Kazman presentaron en 1998 una taxonomía de nueve vistas, decididamente sesgadas hacia el diseño concreto y la implementación:

1. Estructura de módulo; las unidades son asignaciones de tareas.
2. Estructura lógica o conceptual. Las unidades son abstracciones de los requerimientos funcionales del sistema.
3. Estructura de procesos o de coordinación. Las unidades son procesos o amenazas.
4. Estructura física.
5. Estructura de uso. Las unidades son procedimientos o módulos, vinculados por relaciones de presunción-de-presencia-correcta.
6. Estructura de llamados. Las unidades son usualmente (sub)procedimientos, vinculados por invocaciones o llamados.
7. Flujo de datos. Las unidades son programas o módulos, la relación es de envío de datos.
8. Flujo de control; las unidades son programas, módulos o estados del sistema.

9. Estructura de clases. Las unidades son objetos. Esta taxonomía de vista no coincide con ninguna otra.

La recomendación IEEE Std 1471-2000 procura establecer una base común para la descripción de arquitecturas de software, e implementa para ello tres términos básicos, que son arquitectura, vista y punto de vista. La Arquitectura se define como la organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y con su entorno, y los principios que gobiernan su diseño y evolución. Los elementos que resultan definitorios en la utilidad, costo y riesgo de un sistema son en ocasiones físicos y otras veces lógicos. En otros casos más, son principios permanentes o patrones que generan estructuras organizacionales duraderas. Términos como vista o punto de vista son también centrales. En la recomendación se los utiliza en un sentido ligeramente distinto al del uso común. Aunque reflejan el uso establecido en los estándares y en la investigación de ingeniería, el propósito del estándar es introducir un grado de formalización homogeneizando informalmente la nomenclatura. En dicha nomenclatura, un punto de vista (viewpoint) define un patrón o plantilla (template) para representar un conjunto de incumbencias (concerns) relativo a una arquitectura, mientras que una vista (view) es la representación concreta de un sistema en particular desde una perspectiva unitaria. Un punto de vista permite la formalización de grupos de modelos. Una vista también se compone de modelos, aunque posee también atributos adicionales. Los modelos proporcionan la descripción específica, o contenido, de una arquitectura. Por ejemplo, una vista estructural consistiría de un conjunto de modelos de la estructura del sistema. Los elementos de tales modelos incluirían componentes identificables y sus interfaces, así como interconexiones entre los componentes. La concordancia entre la recomendación de IEEE y el concepto de estilo se establece con claridad en términos del llamado “punto de vista estructural”. Otros puntos de vista reconocidos en la recomendación son el conductual y el de interconexión física. El punto de vista estructural ha sido motivado (afirman los redactores del estándar) por el trabajo en Lenguajes de Descripción Arquitectónica. El punto de vista estructural, dicen, se ha desarrollado en el campo de la Arquitectura de Software desde 1994 y es hoy de amplio uso.

La estrategia de arquitectura de Microsoft define, en consonancia con las conceptualizaciones más generalizadas, cuatro vistas, ocasionalmente llamadas también arquitecturas: Negocios, Aplicación, Información y Tecnología. La vista que aquí interesa es la de la aplicación, que incluye, entre otras cosas:

1. Descripciones de servicios automatizados que dan soporte a los procesos de negocios;
2. Descripciones de las interacciones e interdependencias (interfaces) de los sistemas aplicativos de la organización, y

3. Planes para el desarrollo de nuevas aplicaciones y la revisión de las antiguas, basados en los objetivos de la empresa y la evolución de las plataformas tecnológicas.

Cada arquitectura, a su vez, se articula en vistas también familiares desde los días de OMT*(Object Modeling Technique) que son 1 la Vista Conceptual, cercana a la semántica de negocios y a la percepción de los usuarios no técnicos; 2 la Vista Lógica, que define los componentes funcionales y su relación en el interior de un sistema, en base a la cual los arquitectos construyen modelos de aplicación que representan la perspectiva lógica de la arquitectura de una aplicación; 3 la Vista Física, que es la menos abstracta y que ilustra los componentes específicos de una implementación y sus relaciones.

Vistas que proponen los diferentes frameworks

Como expresa Rich Hilliard, a quien seguimos en este tratamiento, “aunque expresiones tales como múltiples vistas son algo así como el Santo Grial de la Ingeniería de Software, de requerimientos y de sistemas, su estatus en la Arquitectura de Software es bastante más oscuro. Las razones para ello son múltiples. 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 las diferentes vistas. Sin embargo, los arquitectos practicantes las usan de todas maneras, porque simplifican la visualización de sistemas complejos”.

La idea de contemplar un sistema complejo desde múltiples puntos de vista no es nativa de la Arquitectura de Software contemporánea, ni es una invención de Kruchten, sino que se origina por lo menos en la década de 1970, en el trabajo de Douglas Ross sobre análisis estructurado. La motivación para introducir múltiples vistas radica en la separación de incumbencias (separations of concerns). Las vistas se introdujeron como una herramienta conceptual para poder manejar la complejidad de lo que ya por aquel entonces se llamaban artefactos*, tales como especificaciones de requerimientos o modelos de diseño. En las contribuciones más tempranas, las múltiples vistas de un modelo se basaban en perspectivas fijas, puntos de vistas o viewpoints; casi siempre los puntos de vista eran dos, el funcional y el de datos, ninguno de los cuáles aparece en arquitectura.

En la década de 1980, sin embargo, las vistas comenzaron a multiplicarse, al punto que se realizaron surveys interdisciplinarios y se organizaron conferencias específicas sobre la cuestión, como Viewpoints’96. No hay un límite necesario para el número de vistas posibles, ni un procedimiento formal para establecer lo que una vista debe o no abstraer. El estándar IEEE 1471 no delimita el número posible de vistas, ya que se estima que no puede haber acuerdo en ello, pero señala lineamientos para su constitución y considera que un viewpoint es a una vista como una clase es a

un objeto. Hay una “lista corta” de vistas que se usa en los textos generales de Arquitectura de Software y una “lista larga” que gira en torno de UML, el cual especifica nueve clases de diagramas correspondientes a ocho vistas, como se indica en la Tabla 2. Diferentes textos de los mismos autores, por ejemplo en *Object-Oriented Design with Application de* **Grady Booch** y *The Unified Modeling Language Reference* de **Rumbaugh**, utilizan distintas terminologías no conciliadas; vistas y puntos de vista no siempre se caracterizan como conceptos distintos y el uso de la terminología, aún en el interior de cada texto, es informal e inconsistente. Es difícil creer que esto se encuentra “unificado” de alguna manera. Cuando los promotores de UML hablan de arquitectura, a instancias de Kruchten, cambian su modelo de vistas por uno que se refiere no a puntos de perspectiva o a incumbencias de los participantes, sino a niveles de abstracción; pero aún así, como se ha visto, su definición de arquitectura difiere de la definición estándar.

Tabla 2. Vistas y diagramas de UML, basado en Jamers Rumbaugh, 2000

Área	Vista	Diagramas	Conceptos principales
Estructural	Vista estática	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interfaz
	Vista de casos de uso	Diagramas 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

Hilliard expresa en sus textos iniciales la modalidad clásica de la Arquitectura de Software, encarnada en Garlan y Shaw, no habla de vistas en absoluto; la Arquitectura de Software clásica se funda en una vista singular e implícita, de carácter estructural. Muchos arquitectos de la corriente principal evitan hablar de vistas, porque cuando ellas proliferan se hace necesario o bien elaborar lenguajes formales específicos para tratar cada una de ellas, o bien multiplicar irracionalmente las extensiones del lenguaje unificado. Sin duda las vistas son una simplificación conveniente, o más bien un principio de orden; pero su abundancia y sus complicadas relaciones recíprocas generan también nuevos órdenes de complejidad.

Procesos y Metodologías

En los diferentes marcos, las vistas estáticas se corresponden con las perspectivas particulares de los diferentes participantes (stakeholders), mientras que las vistas dinámicas tienen que ver con etapas del proceso, ciclo de vida o metodología, caracterizadas como requerimiento, análisis, diseño (o construcción, o modelado), implementación, integración (prueba de conformidad, testing, evaluación). La terminología, lo mismo que la articulación temporal del proceso o el ciclo, depende de cada marco. Llegando al territorio de la metodología, hay que decir que durante varios años la Arquitectura de Software discurrió sin elaborarlas más que circunstancialmente, como si se estimara compatible con las prácticas establecidas en Ingeniería de Software, cualesquiera fuesen: RUP*, RAD*, RDS*, ARIS*, CMM*. Hoy en día la metodología dominante en la industria es tal vez el Modelo de Madurez de la Capacidad (CMM), aunque el SEI no la considera formalmente como tal.

Desde 1998 y cada vez con mayor intensidad, el SEI y otros organismos comenzaron a elaborar métodos específicos de procesos de ingeniería que sistematizan el rol de la arquitectura en la totalidad del proceso, desde la licitación de requerimientos hasta la terminación. Algunos de esos métodos son Architecture Based Design (ABD), Software Architecture Analysis Method (SAAM), Quality Attribute Workshops (QAW), Quality Attribute-Oriented Software Architecture Design Method (QASAR), Attribute-Driven Design (ADD), Architecture Tradeoff Analysis Method (ATAM), Active

Review for Intermediate Design (ARID), Cost-Benefits Analysis Method (CBAM), Family-Architecture Analysis Method (FAAM), Architecture Level Modifiability Analysis (ALMA), y Software Architecture Comparison Analysis Method (SACAM). Al lado de esos métodos está surgiendo un nutrido conjunto de técnicas de documentación: métodos, técnicas y estudios de casos. Habiendo al menos una docena de procedimientos complejamente engranados entre sí.

En general, la Arquitectura de Software de la corriente principal todavía no se ha expedido en relación con los llamados métodos ágiles. No hay un solamente método, sino una multiplicidad de posturas más o menos radicales y combativas: Extreme Programming, SCRUM, Crystal Methods Framework, Feature-Driven Development, DSDM, Lean Development, Adaptive Software Development, Agile Modeling, Pragmatic Programming. De hecho, la comunidad de los metodólogos, que opera a una cierta distancia de las iniciativas formales de la Arquitectura de Software, se encuentra dividida tras lo que ha sido y sigue siendo "el gran debate metodológico" entre los métodos pesados (o rigurosos) de tipo SEI/CMM por un lado y los métodos ágiles por el otro. Los teóricos de cada bando han sido, entre otros, Tom De Marco, Ed Yourdon y Tim Lister en la facción rigurosa y Ken Orr, Jim Highsmith, y Martin Fowler del lado ágil-extremo, con Philippe Kruchten y RUP buscando establecerse en ambos terrenos.

Ambos grupos operan en el contexto de la crisis del software, que se da por sentada, alegando que es la mentalidad del bando opuesto lo que la ha ocasionado. Los pesados, mirados por los ágiles como formalistas fracasados, enfatizan el modelado, el control y la documentación escrupulosa; los ágiles, acusados por los pesados de hackers irresponsables que pretenden imponer sus juguetes en la empresa, no sólo desprecian los modelos (formales o informales) sino que incluso ocasionalmente ponen en tela de juicio hasta a los propios patrones de diseño.

Abstracción

El concepto de abstracción (que a veces se usa en el sentido del proceso de abstraer, otras para designar una entidad) ha sufrido también diversas acepciones, con un núcleo de significados común. Las diferencias en el uso del concepto de abstracción ayudan también a identificar las diversas corrientes en el seno de la Arquitectura de Software. La definición que proporciona Grady Booch, por ejemplo, revela que el autor identifica la abstracción arquitectónica con el encapsulamiento propio de la tecnología de objetos: "Una abstracción –escribe Booch– denota las características esenciales de un objeto que lo distinguen de otras clases de objeto y provee de este modo delimitaciones conceptuales bien definidas, relativas a la perspectiva del observador". (8)

Este concepto ha sufrido también diversas formulaciones sintácticas, con un núcleo de significados común. En último análisis, la abstracción siempre conlleva una heurística positiva al lado de una

negación. Tanto para la IEEE como para Rumbaugh, Shaw y otros autores, la abstracción consiste en extraer las propiedades esenciales, o identificar los aspectos importantes, o examinar selectivamente ciertos aspectos de un problema, posponiendo o ignorando los detalles menos sustanciales o irrelevantes.

La idea de abstracción forma parte de la pieza conceptual más importante de la Arquitectura de Software, el concepto de estilo; un estilo se identifica a grandes rasgos o, como se dice habitualmente, en un estilo "menos es más". La misma idea prevalece en todos los conceptos y procedimientos que se consideran arquitectónicos. Para Len Bass, Paul Clements y Rick Kazman, si una decisión debe posponerse hasta el momento de tratar las cosas a un bajo nivel, no se trata de una decisión de arquitectura. Clements y Northro sostienen que el trabajo de Garlan y Shaw sobre los estilos arquitectónicos enseña que aunque los programas pueden combinarse de maneras prácticamente infinitas, hay mucho que ganar si se restringe a un conjunto relativamente pequeño de elecciones cuando se trata de cooperación e interacción. Las ventajas incluyen mejor reutilización, mejores análisis, menor tiempo de selección y mayor interoperabilidad .Conceptos como el de estilo, o marcos como MSF[9], revelan su naturaleza arquitectónica en su abstracción y en su generalidad.

Escenarios

Esta es una noción arquitectónica importante y se encuentra en la base de muchos de los métodos de diseño y desarrollo basados en arquitectura, como ALMA, SAAM y ATAM. Hay que ser precavidos y advertir desde el comienzo que lo que habitualmente se traduce como "escenario" no es estrictamente lo que en lengua castellana se designa como tal; la traducción correcta debería ser más bien "guión" o "libreto". La traducción literal del término no hace más que aportar confusión. Desde Kruchten en adelante, se reconoce que los escenarios se dividen en dos categorías: casos de uso (secuencias de responsabilidades) y casos de cambio (modificaciones propuestas al sistema).

Los escenarios han sido básicamente técnicas que se implementan en la licitación de los requerimientos, particularmente en relación a los operadores de sistemas. También se han utilizado escenarios como método para comparar alternativas de diseño. Los escenarios describen una utilización anticipada o deseada del sistema, y típicamente se expresan en una frase. Kazman, Abowd, Bass y Clements proponen también llamarlos viñetas.

Según algunas definiciones, como la de Clements y Northrop, los escenarios son algo así como libretos (en el sentido teatral o cinematográfico del término) correspondientes a las distintas piezas

[9] Microsoft-Solution-Framework. Flexible serie de conceptos, modelos y prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

de funcionalidad de un sistema. Se les considera útiles para analizar una vista determinada o para mostrar la forma en que los elementos de múltiples vistas se relacionan entre sí. Pueden concebirse también como una abstracción de los requerimientos más importantes de un sistema. Los escenarios se describen comúnmente mediante texto en prosa, utilizando lo que se llama un script y a veces se describen mediante dibujos, como por ejemplo diagramas de interacción de objetos. Se acostumbra a utilizar UML (en el contexto de 4+1) no tanto como recurso de modelado que después generará alguna clase de código, sino como instrumento de dibujo más o menos informal; pero los propios manuales de UML y los expertos mundiales en casos de uso (David Anderson, Martin Fowler, Alistair Cockburn) recomiendan desarrollar los escenarios de requerimiento en texto, no en diagramas. Algunos autores estiman que se trata de una herramienta importante para relacionar vistas arquitectónicas, porque recorriendo un escenario puede mostrar las formas en que fragmentos o escenas de esas vistas se corresponden entre sí.

Conclusiones del Capítulo

Anteriormente, en el Estado del Arte, se logró profundizar en los conocimientos básicos de la Arquitectura de Software, ver sus principales estilos, corrientes, las empresas insignes en esta materia, los autores reconocidos y la importancia de su implementación en la actualidad.

En lo adelante se adentrarán en el desarrollo de este trabajo, podrán ver en cuáles disciplinas se desglosa y por qué, dependiendo de la magnitud del proyecto, y las actividades que se deben llevar a cabo en cada una. También se verá la importancia del Rol del Arquitecto de Software, mencionando algunas definiciones de otros autores y describiendo las actividades que realiza, de acuerdo al área en que se encuentre. Posteriormente se hará una propuesta de Expediente de Arquitectura, para la organización de la documentación que se genera. Y por último se abordará lo que son las Pruebas de Concepto, que les darán una forma de medir la calidad de la arquitectura trazada, todo esto desde la experiencia de varios proyectos en especial la del ERP-Cuba.

Capítulo 2. Vistas (Disciplinas) de la Arquitectura de Software.

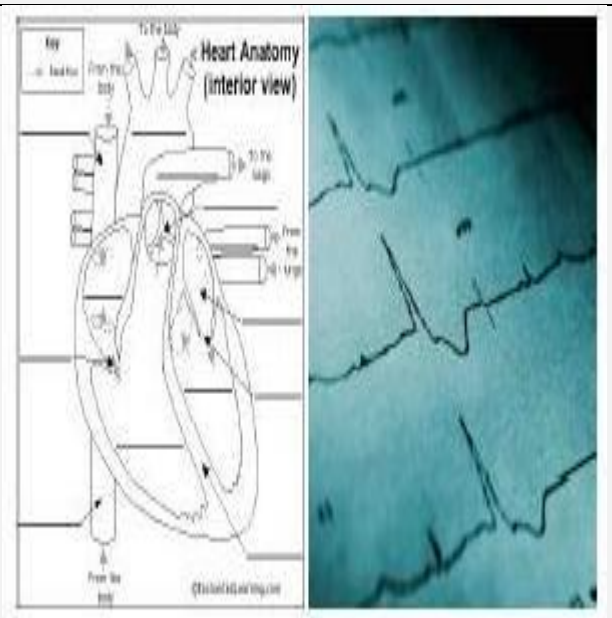
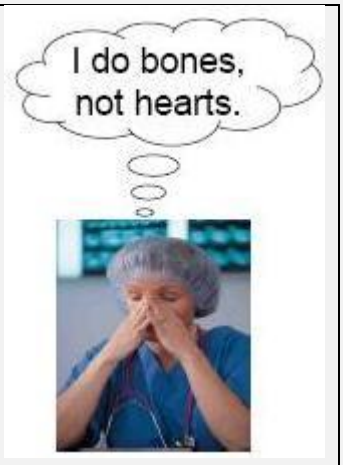
Introducción del Capítulo

Una vista o disciplina de la Arquitectura de Software es una temática desde la que se enfoca el proyecto, ya sea desde el punto de vista de los datos, integración, etc. En este capítulo se proponen las diferentes vistas de la Arquitectura de Software, y de ellas sus conceptos básicos, qué actividades se realizan en cada una de ellas, etc.

Para separar las vistas se debe tener en cuenta todo el trabajo que se lleva a cabo a la hora de estructurar una arquitectura, tratando de no dejar ningún contenido fuera de alguna de ellas. Esto como es lógico no es una labor nada sencilla, y mientras más se crece en los conocimientos arquitectónicos más complejo y engorroso se vuelve, generando muchas dudas sobre dónde va un contenido, si es suficiente con integrarlo a alguna vista ya propuesta o si es indispensable crear una nueva vista solo para él. De ahí que surge esta propuesta, que trata de agrupar todos estos contenidos en las vistas descritas a continuación.

- La metáfora de David Garlan

Tabla 3. La metáfora de David Garlam

 Estas vistas son necesarias para un cardiólogo		 ¿Pero estas mismas vistas funcionarán para un ortopédico?

Esta metáfora ilustra claramente la necesidad de separar las vistas de acuerdo a lo que se necesite describir.

Son muchísimas las clasificaciones en las que se ven envueltas las disciplinas de la Arquitectura de Software. En el capítulo uno se hizo un breve recuento de todas esas corrientes. A continuación se profundizará un poco más en el tema dando las principales clasificaciones y qué se hace en cada una de esas vistas propuestas.

Clasificaciones de vistas:

David Parnas (´74) las estructura en tres grupos:

- Estructura de módulos: es parte de o comparte el mismo secreto que la asignación de trabajo.
- Estructura de uso: depende de la corrección de programas.
- Estructura de procesos: brinda trabajo computacional a procesos.

Perry y Wolf (´94)

Reconocen la necesidad de vistas que enfatizan ciertos aspectos arquitectónicos útiles para distintas audiencias o para diferentes propósitos.

Siemens Corporate Research (´95)

- Vista conceptual: principales elementos de diseño y su interrelación.
- Vista de módulos: estructura funcional y de capas.
- Vista de ejecución: estructura dinámica.
- Vista de código: organización de código fuente, binarios y bibliotecas en el ambiente de desarrollo.

Libro: “Software Systems Architecture” por Nick Rozanski y Eóin Woods

- Vista funcional
- Vista de información

- Vista de concurrencia
- Vista de desarrollo
- Vista de despliegue
- Vista operacional

SEI (´02): Racionalización de Vistas

Categorizan las vistas en “ViewTypes”.

- *Viewtypes:* definen los tipos de elementos y los tipos de relaciones usados para una descripción desde una perspectiva particular.
- ViewType Modular: ¿Cómo está el sistema estructurado como conjunto de unidades de implementación?
- ViewType Componente y Conector: Como un conjunto de elementos que tienen comportamiento e interacción en tiempo de ejecución.
- ViewType Asignación: ¿Cómo se relaciona el software con elementos que no son software?

Propuestas de Vistas Microsoft

- Vista Conceptual

Es usada para definir los requerimientos funcionales y la visión que los usuarios del negocio tienen de la aplicación y describir el modelo de negocio que la arquitectura debe cubrir. Esta vista muestra los subsistemas y módulos en los que se divide la aplicación y la funcionalidad que brinda dentro de cada uno de ellos.

Casos de Uso, Diagramas de Actividad, Procesos de Negocio Entidades del Negocio, etc.

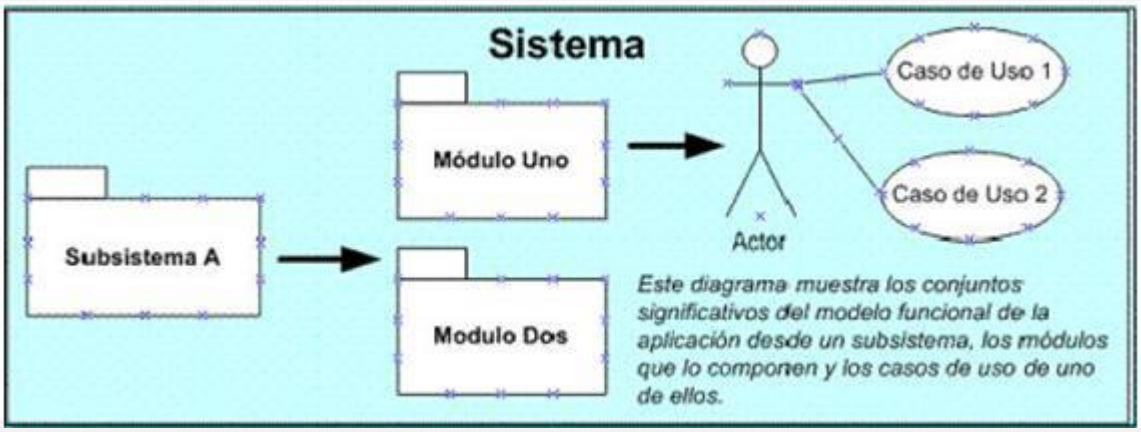


Ilustración 1. Vista Conceptual

- Vista Lógica

Muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución. Los arquitectos crean modelos de diseño de la aplicación, los cuáles son vistas lógicas del modelo funcional y que describen la solución.

Realización de los Casos de Uso, subsistemas, paquetes y clases de los casos de uso más significativos arquitectónicamente.

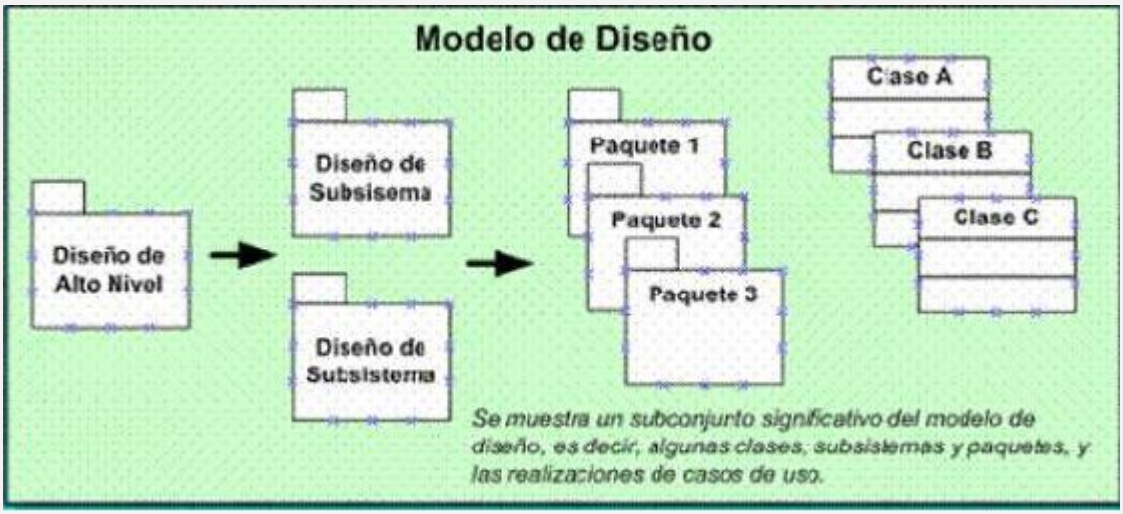


Ilustración 2. Modelo De Diseño

En la actualidad, uno de los patrones de diseño más utilizado para cualquier tipo aplicaciones es el de Capas (Layers en inglés) donde, básicamente, se dividen los elementos de diseño en paquetes de Interfaz de Usuario, Lógica de Negocio y Acceso a Datos y Servicios.

Diagrama de Paquetes.

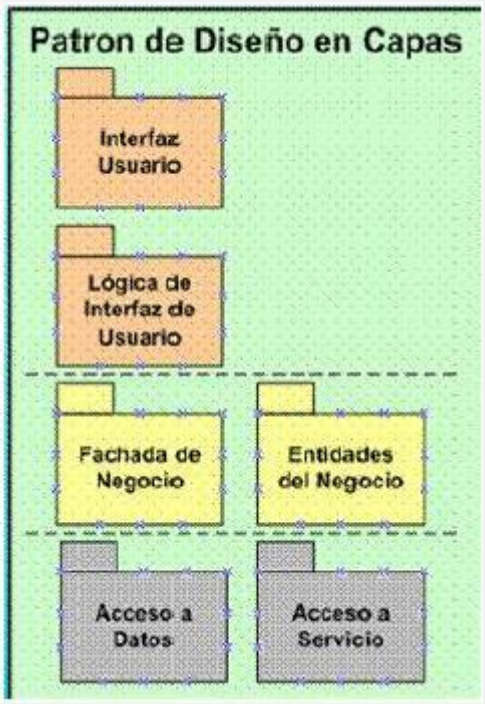


Ilustración 3. Diagrama de Paquetes

- Vista Física

Ilustra la distribución del procesamiento entre los distintos equipos que conforman la solución, incluyendo los servicios y procesos de base. Los elementos definidos en la vista física se "mapean" a componentes de software (servicios, procesos, etc.) o de hardware que definen más precisamente cómo se ejecutará la solución.

Diagrama de Despliegue.

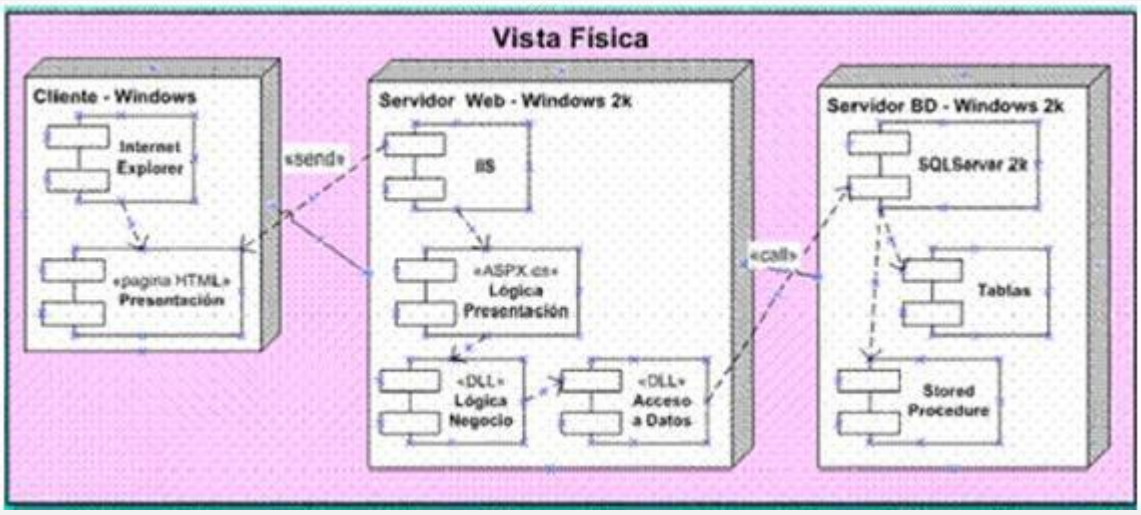


Ilustración 4. Vista Física

- Vista Implementación

Describe cómo se implementan los componentes físicos mostrados en vista de distribución agrupándolos en subsistemas organizados en capas y jerarquías, ilustra, además las dependencias entre éstos. Básicamente, se describe el mapeo desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.

Diagrama de Componentes

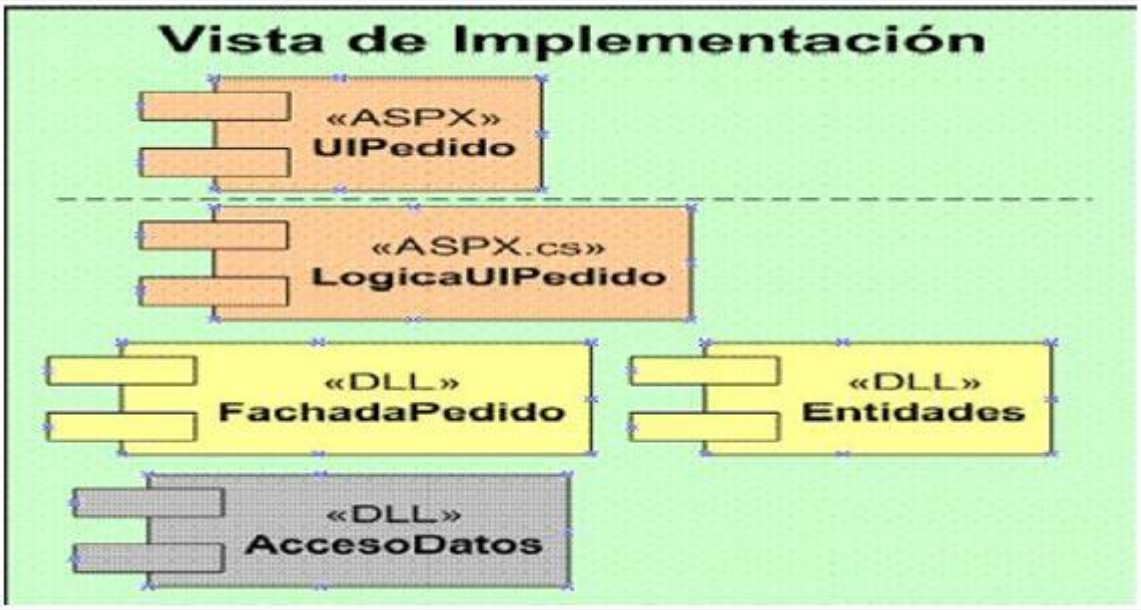


Ilustración 5. Vista de Implementación

Propuestas de Arquitectura 4+1 Vistas de Kruchten (RUP)

- Vista lógica: requerimientos de comportamiento, mecanismos comunes de diseño (basada en diagramas de objetos y clases).
- Vista de procesos: distribución, integridad, tolerancia a fallas (basada en describir una red lógica de programas que se comunican).
- Vista de desarrollo o implementación: rehúso, portabilidad, asignación de requerimientos y trabajo de equipos. Organización del software en el ambiente de desarrollo.
- Vista física o de despliegue: disponibilidad, confiabilidad, performance, escalabilidad. Mapea elementos de las otras vistas a nodos de procesamiento.
- Vista de Casos de Uso o Escenarios: definición de procesos, agrupamiento en paquetes.

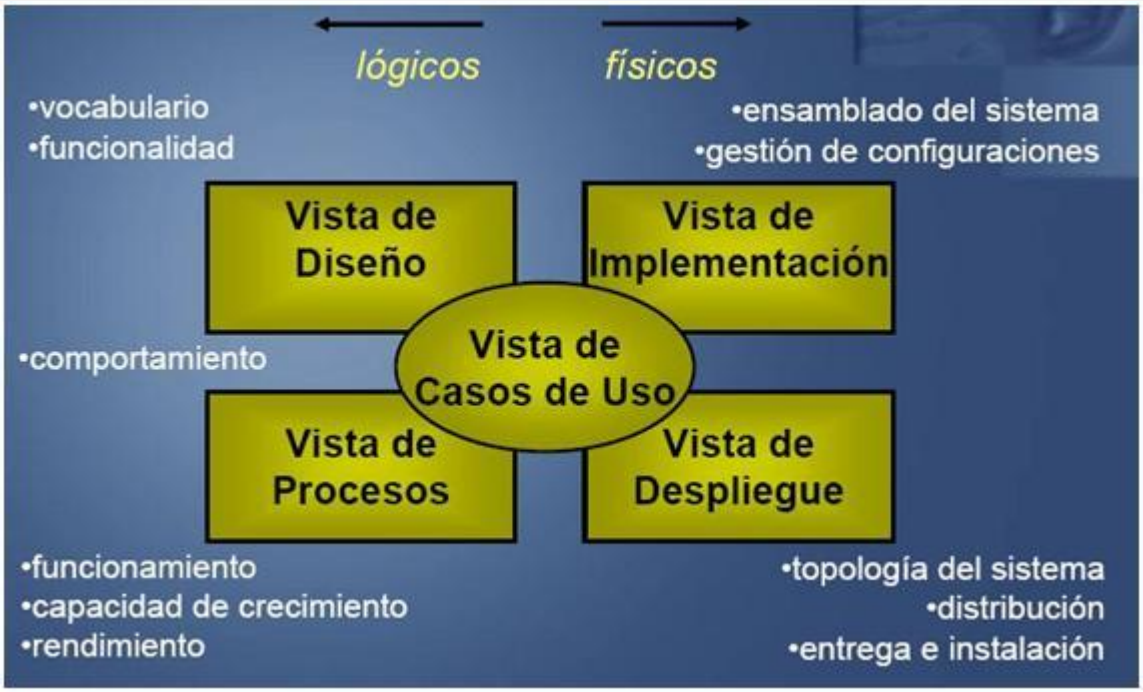


Ilustración 6. 4+1 Vistas de RUP

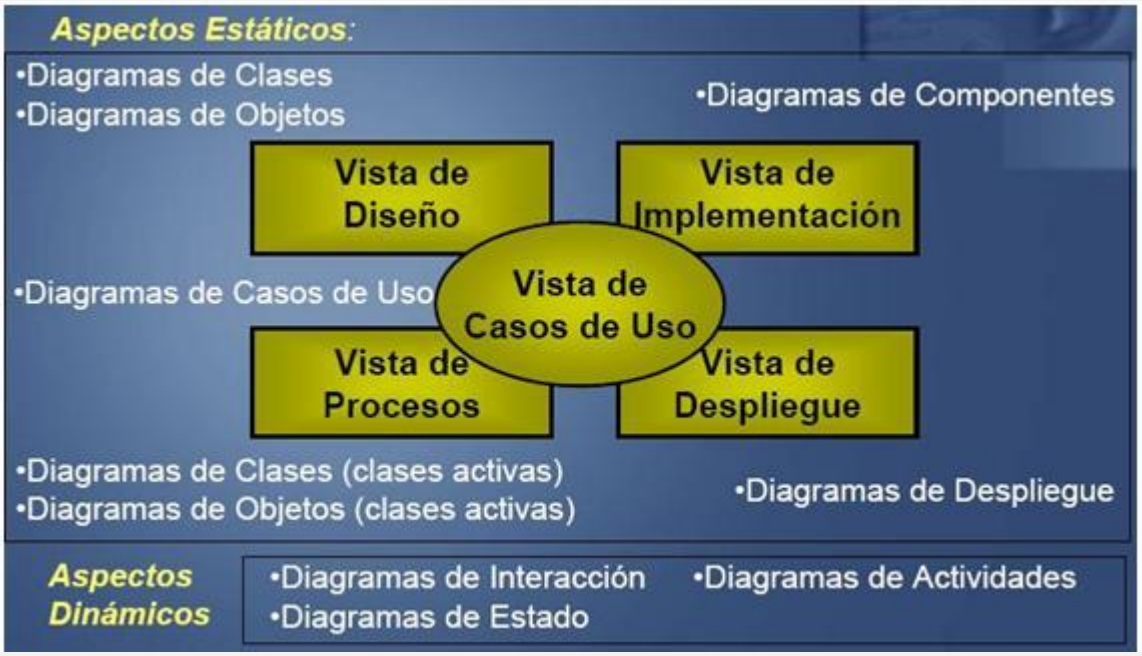
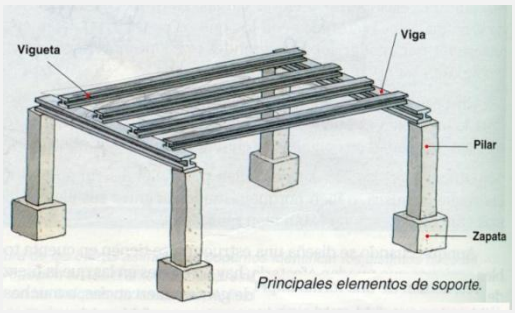


Ilustración 7. 4+1 Vistas Aspectos Estáticos

Hasta aquí se han mostrado las propuestas de diferentes autores para clasificar las vistas que deben conformar la disciplina de la Arquitectura de Software. A continuación se darán las propias definiciones de los autores del trabajo, resultantes de una larga investigación y basadas en la experiencia de varios proyectos. Las clasificaciones que se proponen en este trabajo están dadas por la necesidad de agrupar los elementos que se quedan fuera de las vistas propuestas en una única y global clasificación. Por ejemplo se vieron todos los contenidos que engloba RUP en sus 4+1 vistas, pero descuida los temas referidos a Datos, seguridad, presentación y sistema. **David Parnas**, trata elementos a los que Siemens Research hace adiciones, pero a ambos les falta los temas que aborda el **SEI** (vistos anteriormente), sin embargo a este le falta los aspectos relacionados con la presentación, datos y seguridad. Por su parte Perry y Wolf proponen vistas que enfatizan ciertos aspectos arquitectónicos útiles para distintas audiencias o para diferentes propósitos, propuesta que se queda demasiado general. En el libro “Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives” de Nick Rozanski y Eóin Woods, abordan 5 vistas que abarcan un amplio contenido, pero no tratan las disciplinas de seguridad e integración. De ahí que se propongan estas vistas que se verán a continuación, y en las que el contenido que se encierra se trató fuera el más abarcador posible y estuviera bien desglosado y explicado en las vistas respectivas.

Para que se entiendan mejor las vistas de la Arquitectura de Software y el enfoque que se le dará a cada una en este trabajo, se hará una analogía con la arquitectura tradicional y sus diferentes "vistas".

Tabla 4. Analogía entre arquitecturas

Arquitectura tradicional	Representación	Arquitectura de Software
Elementos de carga, base del edificio. Parte más importante en la construcción.	**Ilustración 8. Elemento de carga** 	Vista Tecnológica: es la base del software, propicia los elementos necesarios para crear el producto.
Las divisiones que tendrá el edificio, las habitaciones, los espacios, en fin las partes en las que estará dividido.	**Ilustración 9. Partes de un edificio** 	Vista de Sistema: propone las partes del software: componentes, conectores…
Los elementos de integración del edificio, como ladrillos, cabillas, todo lo que permita que sus partes queden unidas.	**Ilustración 10. Elementos de integración**	Vista de Integración: identifica todos los aspectos de integración del software.

		
La seguridad de las puertas, ventanas, el acceso al edificio, seguridad de la estructura, etc.	Ilustración 11. Seguridad 	Vista de seguridad: chequea e implementa todos los aspectos relacionados con el acceso a la aplicación, la modificación, lectura o eliminación de la información, etc.
A la hora de construir un edif. hay que tener en cuenta el acabado, la pintura, la arquitectura, la presentación, en fin la imagen del edificio.	Ilustración 12. Presentación 	Vista de presentación: como luce el software, cuáles son los colores que lleva la aplicación, como son los botones, los links…
Todo el personal que vive en el edificio, los productos que hay en el (mesas, camas, sillas), todo se debe tener contabilizado.	Ilustración 13. Datos	Vista de datos: es donde se modela todo lo persistente y relacionado a la BD.

2.1 Vista de la Arquitectura de Sistema.

Arquitectura de sistema, definiciones teóricas

La Arquitectura de Sistema es una de las disciplinas más complejas dentro de la Arquitectura de Software, responsable de definir correctamente cohesionados, acoplados e interrelacionados los elementos computacionales del producto, las principales interacciones, los conectores y las configuraciones a asumir por los elementos computacionales en función de los elementos del negocio que los mismos abstraen. La vista de sistema de la Arquitectura de Software, representa una proyección simétrica de alto nivel, de los procesos de negocio o arquitectura de negocio que se trabaja, expresada en elementos, conectores, configuraciones y restricciones.

Primeramente se abordarán los principios de empaquetamiento del diseño arquitectónico de un sistema de software, estos tienen un alto impacto en el diseño de la solución, ya que esta actividad está asociada con los niveles de colaboración de los elementos computacionales, los principios de reutilización y los niveles de abstracción y encapsulamiento en función del problema que se modela. Una adecuada selección de los mismos suele implicar mejor organización del equipo de desarrollo y paralelización de las tareas de implementación, más facilidad en los procesos de Gestión de Configuración de Software y una granulación adecuada de las partes del software con implicación positiva en la gestión de la integración continua y el propio mantenimiento del sistema. Después de aplicar estos niveles de empaquetamiento se tendrá el negocio desglosado en partes para luego proceder a tratar los temas de categorías arquitectónicas que son las diferentes agrupaciones de componentes que se realizan para abstraer la lógica de un negocio y mejorar la organización e

implementación de un sistema, desglosando estos hasta el nivel más básico. Y finalmente se tratarán las diferentes actividades que se realizan en esta Vista de Sistema, uno de los temas más importantes y polémicos de esta disciplina. Otros elementos a tener en consideración durante los procesos de empaquetamiento son las colaboraciones entre los elementos de software, principios de integración y los niveles de accesos a recursos o componentes horizontales.

Niveles de empaquetamiento de los elementos arquitectónicos.

Por la importancia que presentan los patrones de empaquetamiento o agrupación de las soluciones arquitectónicas, en este trabajo se propone un patrón recursivo por definición, guiado por los niveles de abstracción de cada agrupación, a partir de una proyección simétrica de la arquitectura o procesos del negocio identificado por el grupo de análisis; el mismo cuenta con cuatro niveles, que son responsabilidad de la Arquitectura de Sistema.

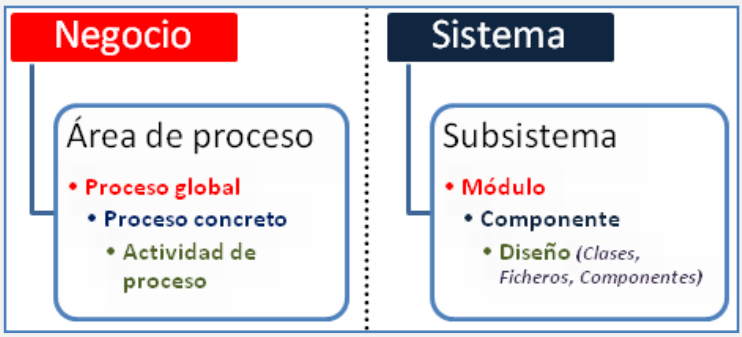


Ilustración 14. Representación de la proyección simétrica de la arquitectura de negocio en arquitectura de sistema

Nivel Sistema. El nivel de sistema es el TODO. Está compuesto por todos los subsistemas, y es la vista global del producto.

Nivel Subsistema. Corresponde con la abstracción de las áreas de proceso del negocio, que posee objetos propios y operaciones asociadas a esos objetos. Tiene implicación en el diseño de instalación de la solución, está compuesto por módulos u otros subsistemas que a su vez están compuestos igualmente por módulos. La colaboración entre subsistemas será gestionada por el nivel más alto, nivel de sistema.

Nivel Módulo. Este nivel surge para empaquetar conjunto de componentes con funcionalidades similares. Puede contener además de los componentes, otros módulos, según el nivel de complejidad del negocio. La colaboración entre módulos será gestionada por el nivel de subsistema.

Nivel Componente. Corresponde con la abstracción de los procesos concretos contenidos en los procesos generales de las áreas de proceso que se modelan. Está compuesto por los elementos

estructurales del diseño y componentes, según el nivel de complejidad del proceso que se modela, la colaboración entre componentes será gestionada por el nivel más bajo de integración en la dimensión lógica de la integración. Este es el nivel base, y tiene implícito un subnivel de Diseño donde se definen los patrones de diseño a utilizar, así como la auditoría y control del cumplimiento de la taxonomía arquitectónica definida en las distintas áreas de la Arquitectura de Software.

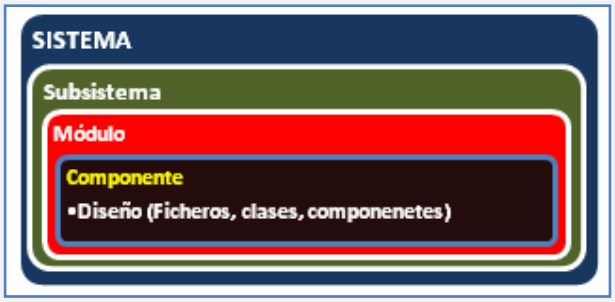


Ilustración 15. Representación de los niveles de empaquetamiento que se proponen

Categorías propuestas para clasificar los elementos arquitectónicos.

Otro elemento a tener en cuenta en el diseño arquitectónico del sistema es la asignación de responsabilidades y la identificación de la colaboración entre los elementos arquitectónicos. La Arquitectura de Sistema además de tener que identificar y diseñar los elementos de software y sus configuraciones producto de la proyección simétrica de la arquitectura de negocio, tiene la responsabilidad de diseñar la integración de los mismos con otros elementos arquitectónicos relacionados con el área de la Arquitectura Tecnológica, Arquitectura de Integración, Arquitectura de Seguridad y la Arquitectura de Presentación fundamentalmente.

Como práctica general un diseño de software de un sistema de gestión presenta cinco grupos de componentes en su generalidad. Un primer grupo relacionado con las implementaciones de la mayor parte del negocio o de los requisitos que se modelan, un segundo grupo asociado con la plataforma tecnológica del desarrollo, elementos que se utilizan para gestionar o dar solución a los elementos computacionales ajenos del negocio, como son las trazas, las excepciones, la concurrencia de información, el caché de datos, la gestión de acceso, la validación de contratos etc., debe recordarse que a diferencia del resto de las industria, en la industria del software por sus características de elementos lógicos, un producto o mercancía, puede formar parte del medio de producción para crear un nuevo producto o mercancía expresado en ceros y unos como el producto que le dio origen, el tercer grupo está asociado a la parametrización* e interoperabilidad de formatos, el cuarto grupo

asociado a los procesos de integración y un quinto grupo asociado a las pruebas unitarias y de integración que garantizan la gestión de integración continua. Teniendo en cuenta estas características identificadas y por la importancia que tiene el desarrollo de habilidades de abstracción en el modelado arquitectónico del sistema, así como su integración con otras disciplinas, los autores del trabajo proponen una formalización de estos cinco grupos en las siguientes categorías arquitectónicas, para que sirvan de guía a los arquitectos de sistema en el diseño de soluciones arquitectónicas de sistemas de software de gestión.

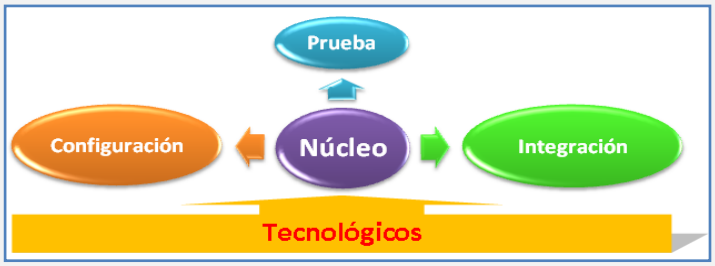


Ilustración 16. Representación de las categorías arquitectónicas que clasifican un elemento arquitectónico

Categoría Núcleo

Representan aquellos componentes con la responsabilidad de abstraer las principales funcionalidades del negocio de la organización, comúnmente agrupan la mayor parte de los conceptos del modelo conceptual, y en términos arquitectónicos agrupan el deseo funcional del sistema que solicita el cliente.

Es característico en ellos un alto impacto en el resto de las decisiones arquitectónicas de los otros grupos de componentes, ya que por contener los procesos núcleos y las principales agrupaciones conceptuales, a partir de sus abstracciones se definen las características de configuración y parametrización que deberá tener el sistema, así como los flujos y procesos de integración de los elementos arquitectónicos.

Categoría Configuración

Representan aquellos componentes con la responsabilidad de abstraer las configuraciones y parametrizaciones estáticas o dinámicas del sistema, así como las conversiones de formato y validaciones de procesos del negocio.

Categoría Integración

Representan aquellos componentes con la responsabilidad de abstraer la estrategia de integración y colaboración del sistema, así como los flujos y subflujos de trabajo. Otra de las responsabilidades arquitectónicas que descansan en este grupo de componentes o elementos arquitectónicos es la interoperabilidad con otras plataformas tecnológicas afines al sistema en cuestión o de otro tipo, como las plataformas ofimáticas, pasarelas bancarias, sistema de gestión de entidades o sistemas legados de la organización que requieren interacción con el sistema en modelación.

Esta categoría asume además los componentes asociados a características horizontales en la arquitectura de sistema, a la implementación de patrones de integración para mejorar la calidad del diseño arquitectónico y la aplicación de los patrones GRAPS y los componentes creados con el objetivo de eliminar lazos de dependencia funcional entre componentes o para implementar políticas de reutilización especificas de la arquitectura de sistema.

Categoría Tecnología

Representan aquellos componentes con la responsabilidad de abstraer características puramente tecnológicas que sirven de base tecnológica al resto de los componentes del sistema, tienen la responsabilidad de abstraer las características no funcionales, con el objetivo de lograr una separación arquitectónica y una mejor especialización entre los diferentes grupos de arquitectos de soluciones, permitiéndose una mejor trazabilidad en la arquitectura del sistema de la arquitectura del negocio. Este grupo de componentes no forman parte en sí de la arquitectura de sistema, aunque la integración de estos con el resto de los grupos de componentes es importante para cada una de las áreas que trabajan, generalmente son constituidos en frameworks tecnológicos que trabajan como especie de mano mágica en la solución arquitectónica y permiten una alta reutilización de las características tecnológicas, además de una alta productividad en el desarrollo, es el grupo de componentes más asociado con la arquitectura tecnológica.

Categoría Prueba

Representan aquellos componentes con la responsabilidad de abstraer las pruebas unitarias y de integración del resto de los componentes de los otros grupos, por el costo y el esfuerzo que requieren generalmente son asociados solamente a los componentes núcleos, con el objetivo de poder maquetar la solución y garantizar que el núcleo funcional es válido, para a partir de una aceptación de la maqueta del producto subcontratar el resto de las características del sistema, los niveles de parametrización, interoperabilidad e integración a incluir en la solución. Los autores del trabajo recomiendan desarrollar un componente prueba por cada componente de alta significancia arquitectónica sin importar el grupo al que pertenezca éste.

2.1.4 Interdisciplinariedad de la arquitectura de sistema

Otra característica distintiva de la Arquitectura de Software es la interacción con otras disciplinas de la misma arquitectura y de otras áreas del conocimiento de la Ingeniería de Software. Entre las más representativas se identifican:

- **Arquitectura de Negocio:** Responsable de describir en procesos el negocio a modelar, garantizándose que el proceso de desarrollo del sistema no tenga como bases requisitos o modelos de procesos errados, que podrían llevar a un diseño no práctico y cuyo mantenimiento una vez implementado hubiese sido más complejo y costoso.
- **Arquitectura de Integración:** Responsable de solucionar las colaboraciones entre componentes, los soportes para la integración arquitectónica entre los componentes y la interoperabilidad de formatos con otros sistemas o plataformas.
- **Arquitectura tecnológica:** Responsable de garantizar un soporte tecnológico para el desarrollo de las configuraciones que propone el resultado de una arquitectura de sistema, así como las bases tecnológicas para los framework especializados de esta o de otras áreas como la integración.
- **Planificación de software:** Usa los elementos de priorización de la arquitectura, así como la clasificación de componentes y la línea base para proyectar el cronograma de desarrollo y las tareas de estimación.
- **Gestión de la configuración del software:** Parte de la línea base definida por la arquitectura de sistema, así como del patrón de agrupamiento o empaquetamiento utilizado.
- **Diseño del sistema:** Parte de los patrones arquitectónicos definidos para cada elemento del software, además de seguir los principios de desarrollo y la taxonomía arquitectónica de la vista de sistema.

Como disciplina la Arquitectura de Sistema hace uso de varios elementos importantes de la Arquitectura de Software como los estilos arquitectónicos, patrones de arquitectura, patrones de diseño, patrones GRAPS, el modelado arquitectónico bien ADL o UML 2.0 entre otros.

La construcción de la Arquitectura de Sistema por su interdisciplinariedad, la cantidad de elementos cognitivos que exige y su implicación en la calidad del producto de software, hacen esta actividad una tarea altamente compleja. Este trabajo propone una guía práctica de actividades que permitan ayudar al desarrollo de la arquitectura de sistema.

Se propone la siguiente guía de actividades a desarrollar en esta Vista de la Arquitectura de Software.

2.1.5 Guía de actividades para el desarrollo de la arquitectura de sistema

1. **Análisis de los artefactos generados en el negocio**

   En el negocio, los analistas deben generar artefactos como el listado de requisitos y la priorización de cada uno, el mapa de procesos y el modelo conceptual. A partir del análisis de estos artefactos es que la arquitectura de sistema se comienza a modelar, ya que todos estos artefactos son los una abstracción del negocio pero más significativa a la arquitectura.

2. **Análisis de los atributos de calidad (Actividad multidisciplinaria)**

   En esta actividad se persigue el análisis de los atributos de calidad. En la arquitectura de sistema es donde se vela porque la implementación de los componentes y con esto el resultado del producto queden acorde a los atributos de calidad, por tanto es en esta vista donde se vela por el Rendimiento, la Usabilidad, Portabilidad, Escalabilidad y Accesibilidad.

3. **Priorización de los requisitos**

   Esta priorización no es la misma que se hace en el análisis, sino, a partir de la que ya se propuso, se profundiza un poco más y se vuelve a priorizar. La actividad tiene como responsabilidad realizar la priorización de los requisitos, con el fin de identificar los de mayor significancia arquitectónica, los cuáles por su impacto en el desarrollo tendrán mayor atención por parte del equipo de desarrollo en la gestión de los componentes.

   $SARF = \mathrm{CD}+\mathrm{CC}+\mathrm{CCP}+\mathrm{CN}$

   $DONDE$:

   SARF: Significancia arquitectónica de un requisito funcional

   CD: Cantidad de dependencias de otro requisito asociado a él

   CC: Cantidad de conceptos

   CCP: Cantidad de conceptos persistentes

   CN: Complejidad del negocio

4. **Agrupamiento de los requisitos funcionales**

   Esta es otra actividad del negocio, y que genera uno de los principales artefactos a partir del cual se identificarán los componentes. La actividad tiene como responsabilidad realizar un agrupamiento semántico de los requisitos funcionales, a partir de una proyección simétrica del negocio, identificándose la asignación de responsabilidades, los niveles de empaquetamiento, y las estructuras capaces de garantizar la mejor reutilización y colaboración posible, esta actividad deja como resultado un agrupamiento semántico que representan los componentes

candidatos de la arquitectura de sistema. No se incluye en esta tarea, la identificación de componentes abstractos asociados a aplicaciones de políticas de integración o reutilización.

5. **Formalización de los componentes**

   La actividad tiene como responsabilidad identificar y diseñar la configuración de los elementos de mayor abstracción de la Arquitectura de Sistema (Subsistemas, Módulos y componentes más generales), parte del resultado obtenido en la actividad número seis. En esta actividad se identifican los estilos arquitectónicos y patrones arquitectónicos propicios según el problema que se modela. Se realiza una propuesta candidata de los patrones de diseño a utilizar en los principales elementos (Componentes) identificados, se revisa el cumplimiento de los patrones GRAPS en el diseño arquitectónico propuesto. Se realiza además la identificación de las entradas y salidas de cada elemento, las dependencias e integraciones, y los procesos de notificación a realizar en las principales acciones de colaboración para que sean posteriormente gestionadas y desarrolladas por el equipo de Arquitectura de Integración.

6. **Iteración de la formalización de los componentes**

   En esta actividad se van analizando todos los componentes identificados por cada subsistema. Al concluir se hace otra revisión, y otra, tantas como sea necesario hasta llegar a una aproximación lo más exacta posible y que de cómo resultado además los criterios necesarios para identificar los componentes horizontales de la arquitectura, actividad que le continua a ésta.

7. **Identificación de los componentes horizontales de la arquitectura**

   Nota: Uno de los elementos que mas complejizan el diseño arquitectónico, por el grado de integración y acoplamiento que genera en los elementos arquitectónicos, son los requisitos o procesos de negocio horizontales, actividades que son repetidas con alta frecuencia en la mayoría de las áreas de proceso que se modelan y que una vez proyectados en el diseño arquitectónico, resultan funcionalidades de alta incidencia en la colaboración y de elevada frecuencia de solicitud en la ejecución del sistema, por consiguiente, con incidencias en los procesos de cómputo y en el rendimiento de la solución, además de los problemas de reutilización y asignación de responsabilidades, que pueden resultar en diseños de soluciones con problemas de cohesión y acoplamiento o reprogramación de código que siempre afecta el mantenimiento y la reutilización.

   La actividad tiene como responsabilidad identificar y formalizar en elementos arquitectónicos las principales funcionalidades horizontales para todo el diseño arquitectónico, ya bien sean

resultado de las actividades de procesos de negocio horizontales o de abstracciones arquitectónicas relacionadas con la transformación, compatibilización de formatos, aplicaciones de patrones o reutilización de algoritmos y validaciones. Es preciso en esta actividad de la guía práctica tener en cuenta el uso de los patrones GRAPS y de integración. Los elementos arquitectónicos finalmente identificados y formalizados como resultado de esta actividad serán entregados al equipo de arquitectura de integración, responsables de determinar la solución definitiva, además de gestionar los procesos de integración de las mismas al resto de la arquitectura, y realizar las pruebas unitarias y de integración correspondientes.

En este trabajo se propone crear un artefacto que sea una especie de libro Excel, que registre en una hoja la siguiente información.

- Componente
- Descripción
- Responsabilidad arquitectónica
- Patrón(es) que implementa
- Lista de contratos que provee
- Lista de Subsistema/Módulo/Componente que referencia a alguno(s) de los contratos que el componente agrupa
- Evaluación del grupo de Arquitectura de Sistema (Cómo resolverlo, cómo reutilizarlo y si debe ser integrado con otro componente o no).
- Evaluación del grupo de Arquitectura de Integración (Cómo resolverlo, cómo reutilizarlo y si debe ser integrado con otro componente o no).
- Estado del componente
    - Estados posibles
        - -Resuelto (Implementado e integrado a la solución)
        - -Implementación (Se está implementando)
        - -Pendiente (Necesita de más análisis por el equipo de Arquitectura de Integración y Sistema, por tanto no existe una solución final)
        - -Denegado (Arquitectura de Sistema e Integración considera integrado en una solución arquitectónica de concepto global, se propone otra solución)
- Responsable (Quien está al frente de la solución).

8. Identificación de los conceptos globales

Nota: Uno de los elementos que más impacto tienen en el rendimiento, la reutilización y mantenimiento de las soluciones arquitectónicas de software son los conceptos horizontales, los que representan conceptos de alto índice de integración y de alta dependencia de datos referenciada por el resto de los componentes arquitectónicos, los mismos son identificados en la actividad dos de la guía práctica, pero formalizados arquitectónicamente en esta actividad. Es recomendado como buena práctica arquitectónica, diseñar una solución arquitectónica que permita reutilizar la invocación y cálculo de los mismos, así como la integración de estos elementos de reutilización de conceptos globales con otros elementos tecnológicos, como estructuras de persistencia eficientes y elementos de caché, buscando elevar el rendimiento y la reutilización de las soluciones. Se recomiendan para estos diseños arquitectónicos, los patrones de inyección de dependencia* o especies de localizadores de servicio.

La actividad tiene como responsabilidad identificar y formalizar arquitectónicamente el acceso a los conceptos globales de la arquitectura de sistema, diseñándose para ello el elemento arquitectónico más eficiente según las características de los mismos, así como las variables globales que no son más que datos de mucho uso en la aplicación que no constituyen componentes, pero que son básicos para la integración. Casi siempre se dividen en variables de sesión y variables de aplicación. Es preciso en esta actividad tener en cuenta el uso de los patrones GRAPS y de integración. Los conceptos globales finalmente identificados y formalizados como resultado de esta actividad serán entregados al equipo de Arquitectura de Integración, responsables de determinar la solución definitiva de estas características de la solución, además de gestionar los procesos de integración de las mismas al resto de la arquitectura, y realizar las pruebas unitarias y de integración correspondientes.

El trabajo propone crear un artefacto que sea una especie de libro Excel, que registre en una hoja por cada subsistema la siguiente información.

- Necesidad.
- Subsistema o componente que lo provee.
- Descripción.
- Formato en que se solicita.
- Evaluación del grupo de Arquitectura de Sistema (Cómo resolverlo, cómo reutilizarlo y si debe ser integrado con otro componente o no).
- Evaluación del grupo de Arquitectura de Integración (Cómo resolverlo, cómo reutilizarlo y si debe ser integrado con otro componente o no).
- Estado de la necesidad
    - Estados posibles

- -Resuelto (Implementado e integrado a la solución)
- -Implementación (Se está implementando)
- -Pendiente (Necesita de más análisis por el equipo de Arquitectura de Integración y Sistema, por tanto no existe una solución final)
- -Denegado (Arquitectura de Sistema e Integración considera integrado en una solución arquitectónica de concepto global, se propone otra solución)

- Responsable (Quién está al frente de la solución)

9. Clasificación de los elementos arquitectónicos del Nivel Componente

La actividad tiene como responsabilidad clasificar (según las cinco categorías arquitectónicas que se definen en el trabajo) los elementos correspondientes al Nivel Componente de la Arquitectura de Sistema identificados en las actividades de la guía práctica de la 7 a la 9.

Se proponen 5 categorías específicas asociadas a las responsabilidades arquitectónicas de los componentes que se identifican en un diseño arquitectónico. Esta actividad deja como resultado un conocimiento que permite refinar las abstracciones realizadas en el diseño arquitectónico y los niveles de empaquetamiento, además de identificar el impacto e importancia de cada elemento en la arquitectura, su responsabilidad y dependencia de uso arquitectónico. La actividad representa un ejercicio metodológico, para refinar el diseño arquitectónico del sistema, mejorar su estructura organizativa, y las responsabilidades de los elementos arquitectónicos que la componen.

10. Análisis de reutilización

Nota: Una de las actividades que más incidencias tiene en que los desarrollos de software no se ajusten a la planificación y por consiguiente se considere un proyecto ineficiente es la ausencia de políticas de reutilización, y la ausencia de repositorios de componentes. Una mala práctica generalizada en el desarrollo de software es la falta de cultura de reutilización y parametrización de elementos de software, la mayoría de los desarrolladores se concentran en reprogramar cada requisito de un sistema, la ausencia de una política de desarrollo orientada a reutilización y componentes, o consumo y orquestación de servicios, dificulta incluso la reutilización de implementaciones y diseños propios del mismo equipo de desarrollo. El análisis del índice de reutilización facilita al equipo de desarrollo contar con información importante para los procesos de planificación y estimación, conocer de antemano las necesidades de desarrollos asociados a la parametrización e integración de los

componentes adquiridos al resto de los componentes del sistema, y es un indicador a tener en cuenta en el cálculo de la significancia arquitectónica de un componente como se verá en próximas actividades de esta Guía de actividades para el Desarrollo de Arquitectura de Sistema.

La actividad tiene como responsabilidad realizar el análisis de reutilización de los elementos arquitectónicos del sistema y las bases tecnológicas acumuladas en los repositorios de componentes COTS[10] de reutilización. La misma se inicia identificando las áreas de reutilización y la formalización de los componentes de la Arquitectura de Sistema en componentes COTS. Se calcula el índice de reutilización de cada componente o elemento de software a partir de las capacidades de extensión, mantenimiento, parametrización y adquisición del componente. El trabajo propone el siguiente instrumento metodológico para a partir de un cálculo sencillo calcular el índice de reutilización de cada elemento de software.

Tabla 5. Calcular el índice de reutilización

Componente	Extensión	Mantenimiento	Parametrización	Adquisición	Índice reutilización
Componente 1	A	B	C	D	IR = (A + B + C + D)/4

Los valores tanto para cada elemento como para el índice de reutilización resultante cercano a cero, representan una tendencia de reutilización casi nula y por consiguientes altos esfuerzos, un valor cercano a diez representa un índice de reutilización muy favorable que no requiere prácticamente esfuerzo por el equipo de desarrollo.

El umbral cuantitativo y su significado semántico asociado de cada criterio se reflejan en la siguiente tabla:

Tabla 6. Umbral cuantitativo

Extensión	
10	Se reutiliza completamente el componente núcleo
8	Basta con implementar el patrón Adapter, no existe necesidad de modificar las interfaces del componente base, pues el formato coincide
6	Implica transformaciones de formato
4	Implica modificar el código fuente para poder ser extendido
2	Implica modificar el código, no se conoce cómo está hecho, requiere estudiar el componente, para poder ser entendido
0	No reutilizable
Mantenimiento	

[10] Commercial off-the-shelf*'(COTS),* se refiere a un tipo particular de componente, caracterizado por ``ser de índole comercial, generalmente de bajo coste, que es adquirido, seleccionado, probado, validado e integrado por desarrolladores de un sistema software basado en componentes para satisfacer ciertas necesidades del sistema, a partir de unos requisitos específicos.

10	No existe necesidad de dar mantenimiento al componente
8	El mantenimiento no implica modificaciones importantes, puede resolverse en 2 horas hombre de trabajo
6	Implica modificaciones de interfaz y algunas lógicas internas
4	Implica grandes modificaciones, sólo se reutilizan algunas funcionalidades, su importancia recae en la complejidad y esfuerzo de implementación de las mismas
2	Implica grandes modificaciones, sólo se reutilizan algunas funcionalidades, no se conoce el componente ni se dominan los conocimientos necesarios, necesario entrenar al equipo, su importancia recae en la complejidad y esfuerzo de implementación de las mismas
0	No reutilizable
Parametrización	
10	No requiere parametrización
8	No requiere parametrización
6	No requiere parametrización
4	Requiere parametrización baja
2	Requiere parametrización compleja
0	No reutilizable
Adquisición	
10	Registrado en el repositorio de componentes
8	Se puede adquirir de manera gratuita desde alguna comunidad
6	Requiere adquisición, con documentación
4	Requiere adquisición, con documentación deficiente
2	Requiere adquisición, sin documentación
0	No se puede adquirir

11. Diseño de los componentes

La actividad tiene como objetivo diseñar los componentes más importantes y significativos de la arquitectura. Se realiza una identificación y selección de los patrones de diseño a utilizar.

12. Priorización de los componentes

La actividad tiene como responsabilidad realizar el cálculo de la significación arquitectónica de los componentes del sistema, con ello su impacto en la arquitectura y la prioridad de implementación de los mismos, la información obtenida de esta actividad de la guía sirve de base para decidir qué elementos arquitectónicos incluir en cada línea base según la iteración y el presupuesto del que se disponga para el desarrollo de las mismas.

$$SACS = (\Sigma_{i=n}^{i=0}\mathrm{SARF}+\mathrm{CDC}+\mathrm{CE}+\mathrm{CT}+\mathrm{CC}+\mathrm{CP}+\mathrm{CFC\text{-}IR})$$

$DONDE$:

SACS: Significancia arquitectónica de un componente de sistema

CDC: Cantidad de dependencias de otros componentes asociado a él

CE: Cantidad de entidades

CT: Cantidad de tablas

CC: Cantidad de controladores

CP: Cantidad de patrones

CFC: Cantidad de ficheros de configuración

IR: Índice de reutilización de los componentes

13. Discusión del plan de estimación y planificación del proyecto

La actividad tiene como responsabilidad realizar un análisis de la capacidad técnica del equipo, la disponibilidad tecnológica y productiva y se propone un primer boceto del cronograma de implementación y desarrollo de la línea base.

14. Revisión Integral del diseño

La actividad tiene como responsabilidad el refinamiento de los artefactos generados en la fase de diseño, es una revisión completa y profunda, con el objetivo de llevar a la implementación el diseño lo mejor y más comprensible posible.

15. Diseño de la línea base

Esta actividad define la línea que deben seguir los procesos de análisis, diseño e implementación, evaluando las mejores prácticas y estilos vigentes en la actualidad referentes a la Arquitectura de Software que permita a los desarrolladores y demás involucrados tener una idea clara de lo que se está implementando. A partir del documento Excel de Priorización de los Requisitos y en el Documento de Especificación de la Arquitectura del Sistema se realiza este documento. Este permitirá guiar y evaluar el desarrollo del proyecto, según las normas definidas por la Arquitectura, y definir los elementos de configuración.

16. Revisión integral de la Arquitectura de Sistema

Esta actividad tiene como objetivo realizar una trazabilidad general desde el negocio hasta los componentes y estructuras internas de los mismos. Se chequea que todos los requisitos identificados hayan sido contratados por algún componente, así como se revisan los resultados de la arquitectura de integración.

17. Implementación de componentes.

Esta actividad consiste en que una vez identificado, formalizado y diseñado el componente proceder a la implementación del mismo.

18. Análisis y actualización de la política de reutilización

Optimizar el desarrollo no es sólo cuestión del tiempo de culminación o la calidad con que se termine. La posibilidad de reutilizar es un aspecto fundamental en cualquier empresa. Luego de que se establece en un inicio una política de reutilización, con los componentes hasta ese momento definidos, hay momentos en que cambian los elementos del diseño y del desarrollo, y se hace necesario volver a analizar las políticas de reutilización trazadas y actualizarlas en dependencia de las nuevas necesidades.

19. Re- análisis de la Base Tecnológica

La actividad tiene como responsabilidad el análisis, diseño y revisión de los frameworks, y tecnologías actuales y viejos y lograr una valoración en cuanto a factibilidad.

2.2 Vista de la Arquitectura de Datos

[11]La Arquitectura de Datos, (en lo adelante AD) identifica y precisa las mejores clases de datos que apoyan las funciones del negocio definidas en el modelo de negocios. Es la primera de las arquitecturas a ser concretadas porque la calidad de los datos es el producto básico de la función de la Ingeniería de Software. La arquitectura de datos tiene como objetivo puntualizar los principales tipos y fuentes de datos necesarios para dar soporte a las actividades de la empresa, de manera que sean:

- entendibles por los participantes
- completas y consistentes
- estables

Lo primero que se debe definir es qué es un servidor de Bases de Datos (BD): Un servidor de BD es un sistema bajo arquitectura cliente servidor que proporciona servicios de gestión, administración y

[11] (IX Congreso de Ingeniería de Organización Gijón, 8 y 9 de septiembre de 2005 Arquitectura de Empresa. Visión General. Llanos Cuenca González, Ángel Ortiz Bas, Andrés Boza García Centro de Investigación Gestión en Ingeniería de Producción. Universidad Politécnica de Valencia. llcuenca@cigip.upv.es, aortiz@cigip.upv.es, aboza@cigip.upv.es)

protección de la información a través de conexiones de red, gobernadas por protocolos definidos y a los que acceden los usuarios de modo concurrente a través de aplicaciones clientes.

Esta vista especifica arquitectónicamente los elementos constantes en el Modelo de Datos (MD). Describe una apreciación global del MD y su organización por lo que se refiere a las tablas, vistas y almacenamiento de los procedimientos que proporcionan la persistencia al sistema. También describe la cartografía de clases constantes de la Vista Lógica a la estructura de los datos de la base de datos. Esta vista es opcional, ya que sólo se realiza si la persistencia es un aspecto significante del sistema y el traslado del Modelo de Diseño al Modelo de Datos no se hace automáticamente por el mecanismo de persistencia.

La Vista de Datos abarca 12 temas fundamentales, estos son: Políticas de trabajo, Seguridad de datos, Normas de comentariado, Estándares de nomenclatura, Herramientas, Tipos de datos, Concurrencia, Normalización de árboles, Políticas de indexado, Rendimiento, Mantenimiento de la base de datos y Distribución a nivel de datos.

Guía de actividades de actividades para el desarrollo de la Arquitectura de Datos:

1. **Definir Políticas de Trabajo**

Esta actividad tiene como objetivo establecer un sistema de trabajo para los arquitectos que desempeñan este rol. La organización de forma centralizada da una gran ventaja, porque todos los Arquitectos de Datos se rigen por una misma política, se hacen talleres para exponer los aspectos investigados por cada miembro, reuniones de organización del equipo, etc.

Además de debe gestionar cómo se llevará a cabo el Control de Cambios y dentro de este definirse cómo utilizar algún sistema de control de cambios, cómo salvar los cambios en el repositorio con la mayor frecuencia posible, siempre dar una descripción breve pero útil de los cambios realizados en el log del commit/checkin, organizar el repositorio y actualizar la copia de trabajo desde el repositorio antes de comenzar a realizar cambios importantes en los modelos. Y por último definir una organización para ese repositorio donde se gestionará el control de cambio.

2. **Puntualizar Definiciones**

Siempre se debe contar con la definición de todas las herramientas que se seleccionaron para el trabajo en la BD (Base de Datos), o definir los conceptos de los aspectos principales a tener en cuenta en el área donde se esté trabajando. En esta vista igualmente hay que definir los conceptos principales que se manejan aquí. Por ejemplo, cuál sería el concepto de Base de datos, que tipos de

BD existen y cuáles usas en tu proyecto, cómo se clasifican los Modelos de datos y cuáles conceptos se manejan dentro de ellos, qué es un Sistema Gestor de Base de Datos, etc.

3. Diseñar Base de Datos

En el proceso de diseño de un sistema de software, la actividad de diseñar una base de datos ocupa un lugar importante, este es un proceso independiente que presenta varias etapas para modelar la información. La primera etapa sería construir el Modelo conceptual, dentro del que se construye el Modelo Entidad Relación (MER) y el Modelo relacional o de datos, y se debe también llevar a cabo el proceso de Normalización y Desnormalización.

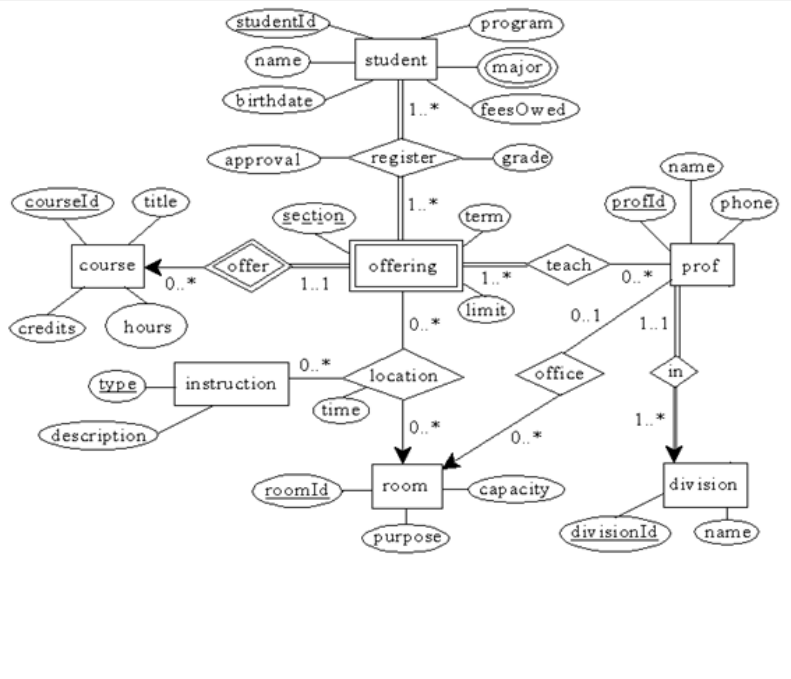


Ilustración 17. Modelo Entidad Relación

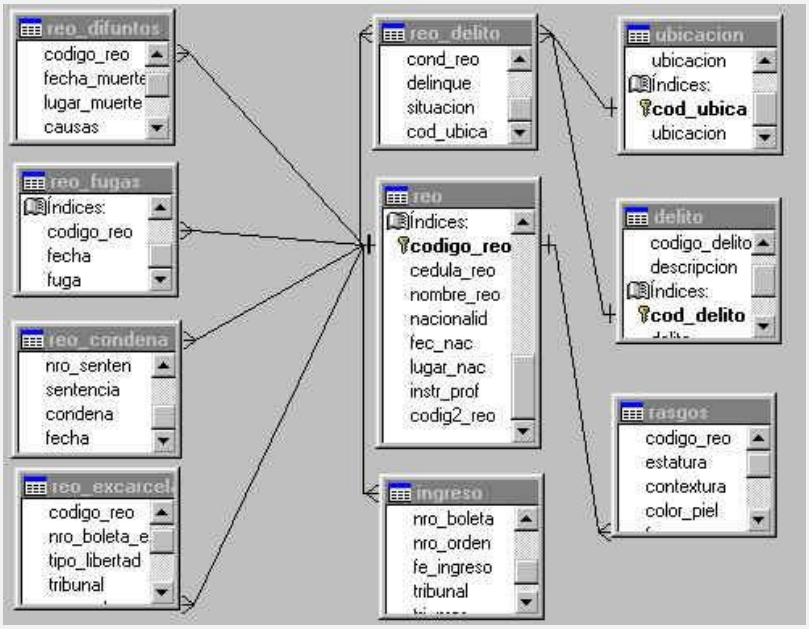


Ilustración 18. Modelo Relacional o Modelo de Datos

4. Definir la seguridad en la BD

Esta actividad se enfoca en qué gestor usar, cómo se protegerá la base datos (BD), cómo se gestionarán los permisos para acceder a esta, cómo utilizar servidores de base de datos centrales para los temas de desarrollo y pruebas, cómo re recuperaran los datos, y cómo se tratarán los temas de respaldos a la BD.

5. Determinar Políticas para el trabajo con la BD

Aquí se deben precisar las políticas para el trabajo con la BD para tener una buena organización y estándar de diseño en la base de datos. Una de las políticas que se definen en esta actividad es cómo será el manejo de los índices. Estos son un tema complejo, ya que si bien aumentan el rendimiento de la base de datos a la hora de realizar consultas, demoran los procesos de inserción, actualización, eliminación. Es una necesidad comentar todo lo que se haga dentro de la base de datos, es decir, establecer las pautas que conlleven a lograr un código más legible y reutilizable, de manera que se pueda aumentar su mantenibilidad a lo largo del tiempo, de ahí que esta sea otra de las políticas a definir aquí. Además se tienen las políticas para la integración entre subsistemas, políticas para el uso de UPDATE y DELETE en CASCADA. Y por último se definen los tipos de datos que deben usarse.

6. Definir Herramientas a utilizar

En esta actividad se deciden cuáles herramientas se usarán tanto a la hora de la creación, la modificación y las relaciones de las tablas como para la administración y el desarrollo de la base de datos, así como los tipos de datos que se usarán en las tablas.

Para la creación, la modificación y las relaciones de las tablas, debe utilizarse siempre una herramienta capaz de mantener los objetos físicos en relación con el modelo lógico, de manera que se logre la documentación. Así que aquí también se identifica qué herramienta de diseño se empleará. Las herramientas de diseño no son las únicas que se necesitan a la hora del trabajo con la BD, también se necesitan herramientas de implementación, de su selección y del conocimiento del arquitecto de datos depende que la construcción de la BD salga con la mayor calidad y robustez posible.

7. Formalizar Definiciones Arquitectónicas

Para el funcionamiento óptimo de la BD se deben evaluar una serie de aspectos y al final tomar las decisiones necesarias al respecto. Estas decisiones deben ser sobre: Creación de las secuencias, donde se decide qué se debe implementar para que la información en la BD esté en diferentes sitios al mismo tiempo y actualizada.

Una misma tupla puede ser accedida por uno o varios clientes al unísono con el fin de realizar determinada acción; en especial a la hora de realizar una modificación. Este problema es llamado Concurrencia, y en esta actividad también se decide cómo solucionarlo. Este tema es muy delicado porque si no se tiene en cuenta puede haber pérdida de datos y puede afectar la integridad de los mismos.

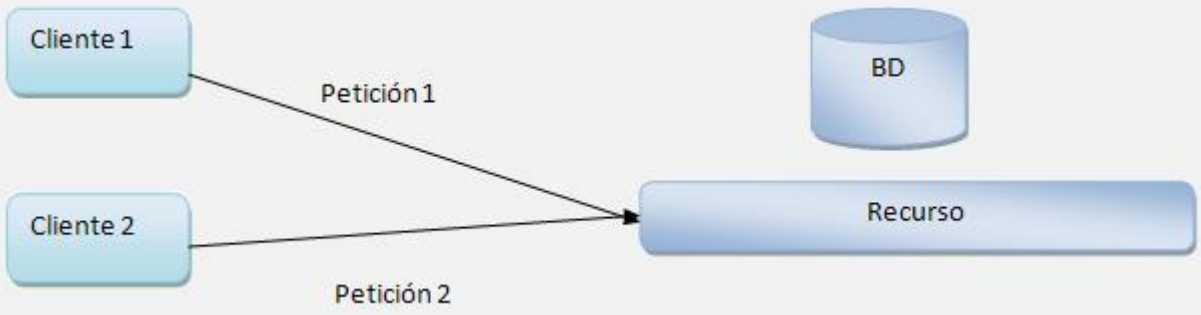


Ilustración 19. Ejemplo de concurrencia

Un ejemplo clásico es la reservación de un asiento de avión. Dos clientes desean tomar el asiento #4 de un determinado vuelo. En sus interfaces tienen el puesto como disponible. Asumiendo que ambos hagan la petición de reservación de manera concurrente qué ocurriría. Aquí se está en presencia del caso base o mejor de los casos, en el cual dos clientes accedan al mismo tiempo. Es evidente que siempre llega una petición primero que la otra, ejecutándose en el orden de llegada. La primera petición realiza la acción y la BD hace una actualización inminente. Entonces la segunda petición puede estar sometida a dos opciones:

- Que modifique el recurso al cual está accediendo, notificando una correcta operación e invalidando la acción de la primera petición.
- (Con un control de concurrencia). Que notifique un error operacional y sea notificado.

Muchos de los sistemas tienen diversas formas de actualizar la capa de presentación, lo que trae consigo la forma de notificar la acción o función realizada. En este ejemplo es evidente que dos personas no puedan tener una misma reservación. Por consecuente la aplicación estaría trabajando bajo un error de concepto.

Solución

Dándole una solución al ejemplo planteado con un componente de concurrencia aplicado. ¿Qué ocurriría? La primera petición se ejecutaría de manera correcta dándole al cliente una notificación operacional. En cuanto a la segunda petición, en el instante de acceder al recurso no se ejecutaría la acción y se le notificaría de la manera requerida o sea al cliente de la segunda petición se le mostraría la interfaz actualizada mostrando el asiento ya reservado.

Otra de las decisiones que se toma aquí es cómo implementar La Normalización de árboles. Esta es necesaria puesto que las tablas de una base de datos correlativa no son jerárquicas, simplemente son listas llanas. Los datos jerárquicos tienen una relación padre-hijo, los cuáles no se representan en la tabla de la base de datos correlativa.

Lo que se muestra aqui es un acercamiento diferente comúnmente referido como modelo de conjunto anidados (encestados). En los modelos de conjuntos anidados se puede buscar la jerarquía de una nueva forma, como contenedores o continentes anidados.
En la figura 16 se muestran categorías, se nota como las categorías padre envuelven a sus hijos.

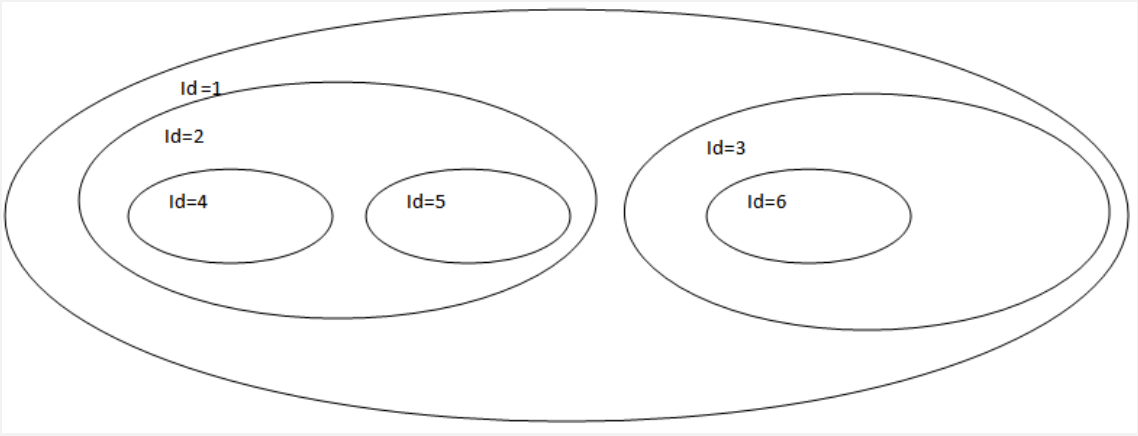


Ilustración 20. Categorías

Concluyendo, el modelado de una estructura de árbol en la base de datos, permite el conocimiento de todos los hijos pertenecientes a un determinado padre y viceversa.

8. Definir Políticas de Indexado

En esta vista se lleva a cabo también la implantación de las **Políticas de Indexado**. Los índices son un sistema especial que utilizan las bases de datos para mejorar su rendimiento global. Son la forma comúnmente utilizada para mejorar el rendimiento de la base de datos, porque permite al servidor de base de datos encontrar y recuperar una fila en específico más rápidamente que si no existiera. Sin embargo los índices deben usarse adecuadamente, pues se agregan en memoria. La presencia de índices en las tablas mejoran considerablemente las operaciones de los datos, siempre, con cuidado con la sobrecarga, porque las modificaciones serán más lentas y por otro lado, las operaciones de actualización e inserción se pueden ver perjudicadas con el hecho de que los índices se tienen que actualizar. Estos pueden ser creados sobre uno o varios campos de una tabla y la creación de un índice único permiten que no se creen duplicaciones de campos.

En esta actividad se definen las estrategias más adecuadas para el indexado. Por ejemplo cómo diseñar el índice, determinar el mejor método de creación, cómo crearlo, etc.

9. Evaluar el rendimiento en la BD

Otra de las actividades que se realizan en esta vista es velar por el rendimiento de la base de datos. La calidad de una BD está dada principalmente por su rendimiento. Cuando se diseña una base de datos, es importante asegurar que realiza todas las operaciones de forma rápida y correcta. Algunos problemas de rendimiento se pueden resolver en el momento en que la base de datos se encuentra

en producción. Sin embargo, otros pueden ser el resultado de un diseño inadecuado y se pueden solucionar mediante el cambio de la estructura y el diseño de la base de datos.

Cuando se diseña e implementa una base de datos, deben identificarse las tablas de gran tamaño y los procesos más complejos que realizará la base de datos. También se debe prestar una atención especial al rendimiento cuando se diseñan estas tablas, además de considerar los efectos que puede tener en el rendimiento el aumento del número de usuarios con acceso a la base de datos.

10. Gestionar el tamaño de la BD

Gestionar el tamaño de la Base de Datos es una de las actividades más importantes a desarrollar puesto que esta influencia en el rendimiento, en la manipulación y en otros aspectos como el almacenamiento. Para lograr reducir el tamaño de una BD se debe establecer un procedimiento que indique los elementos a tener en cuenta para lograr este propósito. Si se controla el tamaño de la base de datos, el rendimiento suele mejorar, con lo que un número mayor de usuarios podrá tener acceso a ella. De manera general, éste se puede ver atacando desde dos frentes:

1. Hardware: configurando y optimizando los recursos hardware de que va a disponer el servidor, como por ejemplo la memoria.
2. Software: utilizando técnicas basadas en software como el uso de índices, optimización de consultas, VACUUM*, etc.

11. Realizar mantenimiento a la Base de Datos

En esta actividad se define cómo se llevará a cabo el mantenimiento de la BD. Las BD deben tener un mantenimiento continuo con el objetivo de borrar ficheros que quedan guardados en memoria. Por ejemplo esto se puede realizar a través del proceso de optimización mediante el VACUUM que se define como el proceso que realiza la limpieza de las bases de datos. Este proceso puede ser ejecutado tanto desde distintos escenarios como configurado desde el sistema operativo o mediante sentencias SQL*, también se configura desde el archivo .conf que acompaña las BD.

Otra de las buenas prácticas para lograr garantizar el rendimiento óptimo de una base de datos es aprovechar la utilidad ANALYZE, para detectar y resolver problemas en su funcionamiento. Si ejecuta ANALYZE regularmente, sobre todo en casos en los que se hable de una base de datos con grandes volúmenes de datos y donde además estos son muy usados, puede detectar pequeños problemas y corregirlos antes de que se agraven. La reconstrucción de los índices o REINDEX, es también una de estas prácticas, ya que puede haber índices incorrectos por problemas en el software o el

hardware, o porque haya que optimizar el índice debido a que tiene páginas que es necesario eliminar. Usando el comando REINDEX, se reconstruye un índice utilizando los datos almacenados en el índice de la tabla. Y al igual que el proceso de VACUUM se puede configurar desde el SO o mediante sentencias SQL.

Otra actividad que entra dentro del mantenimiento a la base de datos es el TUNNING*. Éste implica configurar los procesos de base de datos para que sólo consuman los recursos necesarios sin que se vea afectada la velocidad de respuesta al hacer operaciones en ella. En ocasiones significa configurar la cantidad de bloqueos que se permiten a la vez, otra es determinar la cantidad de semáforos que ocupan los procesos, la cantidad de usuarios permitidos, etc. También el Tunning significa estructurar una base de datos en disco, es decir que los diferentes volúmenes de información estén en varios discos físicos, esto ayuda a que las cabezas de escritura puedan trabajar en varios lados a la vez, si la base de datos está en un sólo disco únicamente puede ser atendida por un cabezal de escritura al mismo tiempo. El Tunning de BD en resumen es configurar los elementos del servicio de base de datos para que su manipulación sea óptima en velocidad y en uso de memoria.

12. Distribuir información a nivel de datos

La Distribución a nivel de datos se propone cuando es un sistema de gran dimensión y complejidad. Esto sería de la siguiente manera, la base de datos general se encontrará en un lugar designado, luego en dependencia de los nodos (organismos, empresas, sociedades, fábricas, en fin cada lugar donde se desplegará el software) que usen el producto en cuestión, se guardará por cada uno de ellos las base de datos correspondiente y así consecutivamente, replicando en todo momento la información desde el nivel más bajo hasta la sede central en la que se encuentre la BD general.

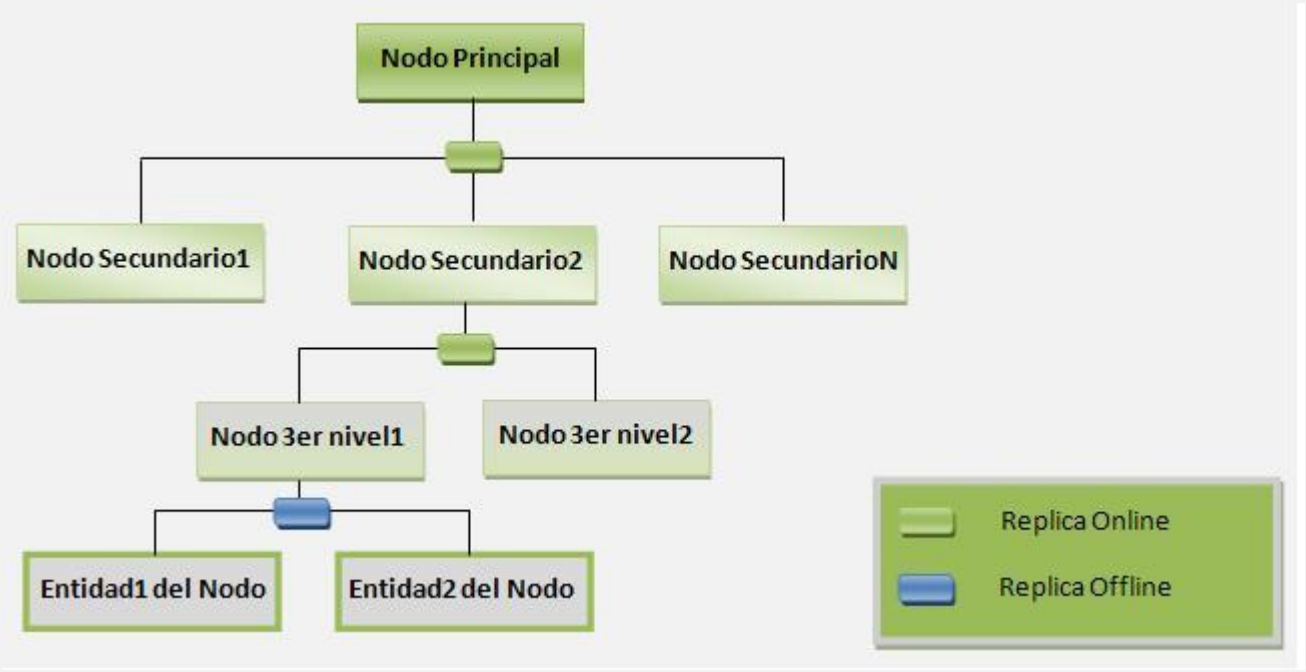


Ilustración 21. Distribución a nivel de datos

2.3 Vista de la Arquitectura de Integración

Integrar es hacer que alguien o algo pase a formar parte de un todo. La integración recoge todos los elementos o aspectos de algo y lo incorpora al ente o a un conjunto de organismos. La Integración en la Arquitectura de Software busca una completa relación del espacio interior con el espacio exterior. Una dualidad que se complementa mutuamente con las características propias de cada ambiente o de cada plataforma operacional en el desarrollo de software. La Arquitectura de Integración persigue la obtención de una forma más eficiente y flexible de combinar recursos, con el objetivo de optimizar operaciones a través y más allá del medio ambiente organizacional. Provee una vista única consolidada a partir de conectores que definen y especifican el comportamiento e interacción entre elementos del negocio (sistemas, subsistemas y componentes).

El objetivo que persigue esta vista, es plasmar detalladamente qué es lo que se hace en ella y cómo se hace. Aquí se analizan los procesos de integración del negocio; se identifican los principales flujos de colaboración o unificación en la arquitectura; se establecen los conceptos más críticos en la integración de sus procesos, de acuerdo al nivel de incidencia en los mismos, partiendo del análisis

de su arquitectura, lo que permitirá conocer las áreas más críticas de integración y por último se analizan entradas y salidas de cada componente identificado por la arquitectura de sistema. Esta actividad permite estar al tanto de los lazos de colaboración entre componentes y clasificarlos según la estrategia de integración identificada por el grupo de arquitectura, construyéndose de esta manera la matriz de integración del negocio.

En este trabajo se expondrán todos estos aspectos de forma suficientemente abarcadora para sentar las bases de cómo construir la Arquitectura de Integración

Guía de actividades para el desarrollo de la Arquitectura de Integración:

1. **Identificar y establecer conectores.**

Esta actividad inicia una vez identificados los componentes de software, con un modelo de los componentes identificados. Se señalan las dependencias internas entre componentes y las externas en caso de existir. Se establecen los contratos que serán utilizados para la interacción entre los componentes y se identifican posibles integraciones con elementos que forman parte de las soluciones verticales del sistema y con variables globales de la solución.

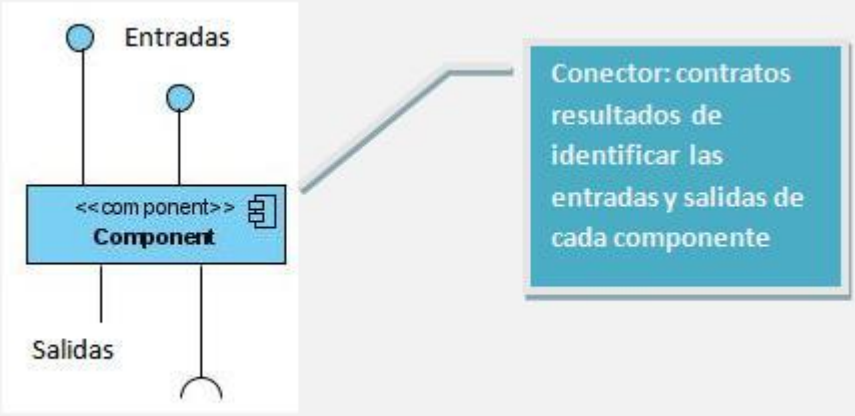


Ilustración 22. Conector

2. **Formalizar conectores.**

Los contratos, son los llamados servicios o procedimientos que solicitará cada componente, son creados en interfaces bien definidas y contienen más de un servicio.

Ilustración 23. Interfaz

3. **Establecer y normalizar la comunicación.**

Define cómo se va llevar a cabo la interacción entre componentes. En general se definen protocolos de integración y las variables de intercambio.

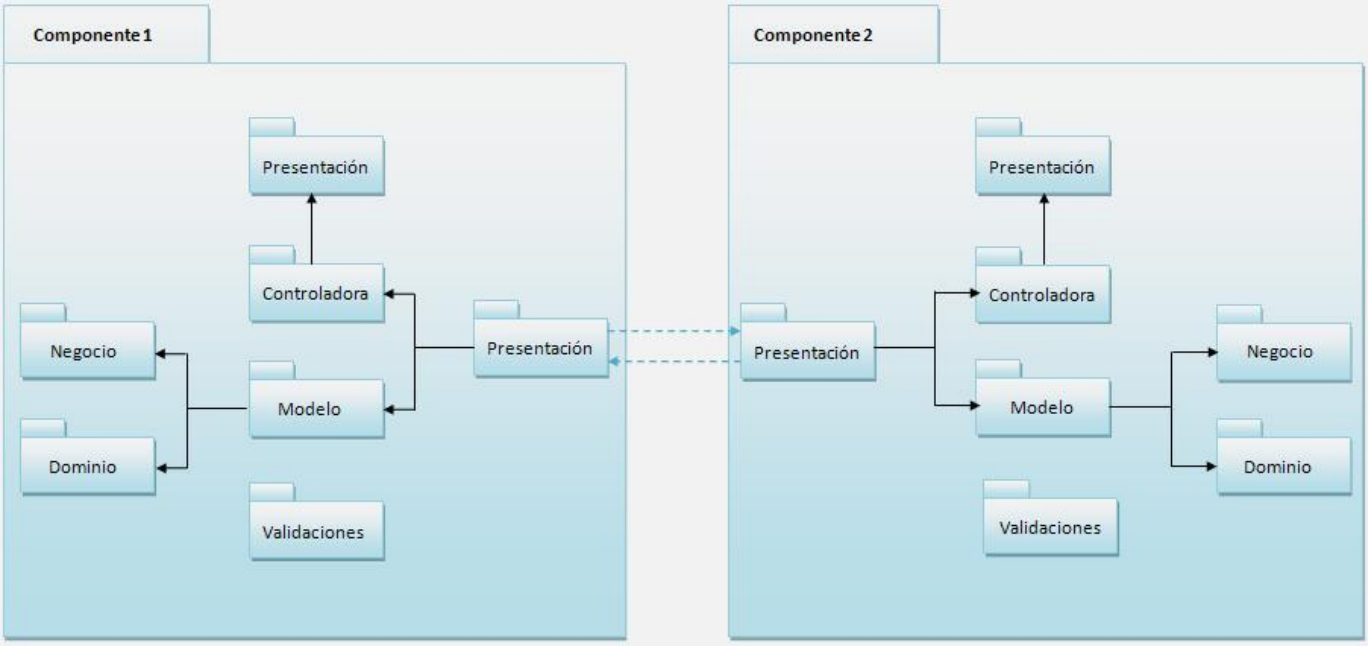
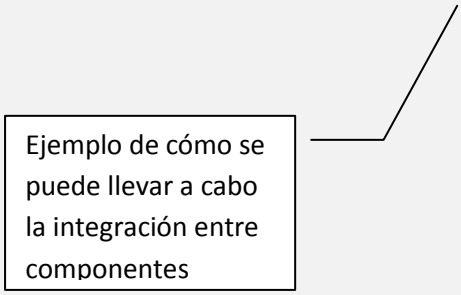


Ilustración 24. Integración entre componentes



4. **Analizar y proponer soluciones de integración con aplicaciones externas.**

El Arquitecto de Integración hace una propuesta de Arquitectura de Integración con otras plataformas de BackOffice o FrontOffice, etc. (Sistemas tales como ERPs*, CRM*). Este arquitecto establece, norma y justifica cada uno de los elementos de integración. Se encarga de documentar, planificar, formalizar y evaluar la integración a todos los niveles y dimensiones.
Estas actividades que se mencionaron anteriormente son de manera más general lo que hace en la Arquitectura de Integración. Ahora se profundiza un poco más, y se pueden señalar otras tareas como:

5. **Análisis de los componentes horizontales.**

Esta actividad tiene como objetivo identificar los patrones de integración y diseño más propicios para el desarrollo de componentes con alta incidencia en la integración, así como identificar los hilos de procesamiento asociados a estos que puedan derivarse en la integración, analizándose la dimensión del buffer para la compartición de memoria, el acceso a recursos concurrentes y las previsibles implicaciones de abrazo fatal[12].

6. **Identificar los flujos de notificación.**

Identificar los flujos existentes en la arquitectura de negocio, cuya trazabilidad a la arquitectura del sistema deberá ser gestionada y controlada por el equipo de integración.

[12] Un conjunto de procesos está en un **abrazo fatal** cuando todos los procesos en ese conjunto están esperando un evento que sólo puede ser causado por otro proceso del mismo conjunto.

7. **Identificar los puntos de acceso a recursos concurrentes.**

Permite realizar el análisis de las características de acceso, la previsión de la concurrencia y abrazos fatales (memoria, datos, ficheros).

8. **Identificar acciones asociadas a flujos de trabajo.**

Identificar las reglas pre y pos condicionales, así como la notificación de eventos y trazas de los mismos, asociadas a las acciones. Con esta actividad se hará un análisis también de los parámetros a comunicar entre actividades y el consumo de buffer de los mismos.

9. **Identificar los formatos y los estándares.**

Esta actividad tiene la responsabilidad de identificar los distintos formatos y estándares a utilizar por la arquitectura, diseñándose los medios de interoperabilidad y transformación.

10. **Formalizar los recursos de integración.**

Esta actividad tiene la responsabilidad de convertir en contratos formales cada elemento asociado a algún flujo de integración identificándose para cada uno:

- Formatos.
- Fuente que lo crea.
- Fuente que lo valida.
- Medio que lo transporta (entre formatos).
- Identificador URI.

11. **Formalizar nodos de integración.**

Aquí se registran los diferentes nodos de integración permitiéndose controlar, monitorear y auditar el desarrollo y prueba de la arquitectura de integración. Se debe tener en cuenta que la integración de un sistema siempre está asociada a un contrato del negocio que son conceptos complejos de mantener y probar.

Nodo de integración

- Elementos asociados (conectores y recursos)
- Contrato:
  - Reglas preposicionales.
  - Reglas post condicionales.
- Formato.
- Notificación.

- Excepción asociada.
- Contrato o requisito del negocio que abstrae elementos o pasos para la formalización de un componente.
- Traza.

Es importante resaltar que los flujos de trabajo no serán registrados como nodos de integración, pues un flujo en si queda formalizado como un nodo de integración.

12. Formalización de los flujos de trabajo estáticos.

Paso

Tabla 7. Formalización de los flujos de trabajo estáticos

Acción inicio-Recurso	Recurso
Precondición: Condición Paso 1.1: Acción (pre y pos condición) Paso n. m: Acción (pre y pos condición)	Formato del recurso
Pos condición: Condición Paso 1.1: Acción (pre y pos condición) Paso n. m: Acción (pre y pos condición)	

13. Formalización de grupo de acciones dinámicas controladas.

Esta actividad tiene la misión de identificar aquellas acciones de integración asociadas a flujos que se pueden conocer de antemano, pero que no se puede determinar su precedencia.

Grupo de acción de FDF[13]

Tabla 8. Grupo de acción de FDF

Acción inicio-Recurso	Recurso
Precondición:	Formato del recurso

[13] Flujo de dependencias funcionales

Condición Paso 1.1: Acción(pre y pos condición) Paso n. m: Acción(pre y pos condición)	
Pos condición: Condición Paso 1.1: Acción (pre y pos condición) Paso n. m: Acción (pre y pos condición)	

14. Identificar niveles de integración.

En esta actividad, después de tener los componentes o subsistemas a integrar, se identifica el nivel según los indicadores que presenta el ambiente a integrar.

15. Planificar la integración.

Esta tarea describe cómo planear el orden de integración de los elementos contenidos en un subsistema de implementación. Se describen los componentes operacionales y se determinan las clases que participan en los escenarios seleccionados. Además, se consideran los subsistemas de implementación que son necesarios para el componente operacional.

De estas actividades, responsabilidades o tareas, se obtienen como resultado una serie de artefactos que son:

- Expediente de la arquitectura de integración.
- Especificación de la arquitectura de integración.
- Soluciones de la arquitectura de integración.
- Decisiones de la arquitectura de integración.
- Listas de chequeos.
- Matriz de integración.
- Plan de integración.
- Formalización de la integración.
- Modelo de componentes integrados.
- Especificación de las interfaces y contratos definidos.

2.4 Vista de la Arquitectura de Seguridad.

Actualmente en muchos de los proyectos de desarrollo de software no existe un área especializada en garantizar la seguridad del entorno de desarrollo, de despliegue y de las aplicaciones propiamente. Por esta causa implementar la seguridad resulta trabajoso ya que cada proyecto o institución la controla de forma diferente, invirtiendo tiempo y cuantiosos recursos humanos y materiales. Esto trae como consecuencia que no se cuente con un entorno seguro de desarrollo y puedan acceder a la información personas no autorizadas. Además, al no establecerse una política estricta en todas las fases del proyecto se pueden cometer errores que conlleven a huecos de seguridad del sistema.

En este sentido se estable un área de seguridad denominada Arquitectura de Seguridad, que es la encargada de establecer toda la política de seguridad a seguir en las diferentes fases o entornos de un sistema. Para esto deben identificar claramente todas las debilidades que puedan ser aprovechadas para un ataque. Estas debilidades deben ser agrupadas para crear soluciones que impidan violaciones de seguridad. Para ello se proponen soluciones como:

- Desarrollar un sistema de seguridad centralizado que brinde estos servicios a todos los proyectos o instituciones que desarrollen software. De esta forma se ahorra tiempo y esfuerzo y además se garantiza la estandarización de los procesos de gestión de seguridad en todos los sistemas que se desarrollen.
- Crear listas de chequeo por cada fase del software donde se controle el cumplimiento de todas las normas de seguridad establecidas.
- Definir la vista de Arquitectura de Seguridad que sirva de guía para el desarrollo y norme todos los aspectos a tener en cuenta en cada fase.

Se puede hablar en este sentido de cuatro aspectos básicos de seguridad: autentificación, confidencialidad, integridad y el no-repudio.

La **autentificación** (o autenticación) es el proceso de verificar formalmente la identidad de las entidades participantes en una comunicación o intercambio de información. Por entidad se entiende tanto personas, como procesos o computadoras.

Existen varias formas de poder autentificarse:

1. Basada en claves.
2. Basada en direcciones.
3. Criptográfica.

De estas tres posibilidades la más segura es la tercera, pues en el caso de las dos primeras es posible que alguien escuche la información enviada y pueden suplantar la identidad del emisor de información.

Desde otro punto de vista se puede hablar de formas de autentificarse, como puede ser a través de la biometría (huellas digitales, retina del ojo, la voz...), por medio de claves, y por último utilizando algo que poseamos, como un certificado digital.

Se llama autentificación fuerte a la que utiliza al menos dos de las tres técnicas mencionadas anteriormente, siendo bastante frecuente el uso de la autentificación biométrica, que como se indicó antes se basa en la identificación de personas por medio de algún atributo físico.

La **autorización** es la parte del sistema que protege los recursos del sistema permitiendo que sólo sean usados por aquellos consumidores a los que se les ha concedido autorización para ello. Los recursos incluyen archivos y otros objetos de dato, programas, dispositivos y funcionalidades provistas por aplicaciones.

La **confidencialidad** es la propiedad de la seguridad que permite mantener en secreto la información y solo los usuarios autorizados pueden manipularla. Igual que antes, los usuarios pueden ser personas, procesos, programas...

Para evitar que personal no autorizado pueda tener acceso a la información transferida y que recorra la Red se utilizan técnicas de encriptación o codificación de datos.

Hay que mantener una cierta coherencia para determinar cuál es el grado de confidencialidad de la información que se está manejando, para así evitar un esfuerzo suplementario a la hora de decodificar una información previamente codificada.

La **integridad** de la información corresponde a lograr que la información transmitida entre dos entidades no sea modificada por un tercero y esto se logra entre otros mecanismos mediante la utilización de firmas digitales.

Mediante una firma digital se codifican los mensajes a transferir, de forma que una función, denominada hash*, calcula un resumen de dicho mensaje y se añade al mismo.

La validación de la integridad del mensaje se realiza aplicándole al original la misma función y comparando el resultado con el resumen que se añadió al final del mismo cuando se calculó por primera vez antes de enviarlo.

Mantener la integridad es importante para verificar que en el tiempo de viaje de la información por la Red entre el sitio emisor y receptor el mensaje no ha sido modificado por personal no autorizado.

Los servicios de **no-repudio** ofrecen una prueba al emisor de que la información fue entregada y una prueba al receptor del origen de la información recibida.

Con este aspecto se consigue que una vez que alguien ha mandado un mensaje no pueda renegar de él, es decir, no pueda negar que es el autor del mensaje.

Independientemente de los aspectos tratados anteriormente, seria bueno a la hora de implementar una Arquitectura de Seguridad, tener en cuenta los siguientes temas:

Administración de perfil

Es la forma de personalización de las aplicaciones de un dominio a nivel de cada usuario, se define como perfil los datos únicos de cada recurso dentro del sistema que define el comportamiento del mismo ante las entrada emitidas por este recurso y las salidas entregadas por el (los) subsistema, esto garantizaría un sin número de ventajas tanto de usabilidad como de configurabilidad a la solución en cuestión.

Administración de conexiones

Bajo este acápite se agrupan todos los procesos para la gestión de las conexiones a bases de datos de un sistema determinado ubicado en un servidor de bases de datos definido también como un parámetro configurable, así como el gestor en uso.

Auditoría

Es el examen objetivo y sistemático de las operaciones financieras y administrativas de una entidad, practicado con posterioridad a su ejecución y para su evaluación.

Se hace necesaria entonces la implementación de un mecanismo que garantice tanto el mantenimiento de este entorno seguro como los demás puntos clave de la seguridad: confidencialidad, autenticación, autorización, integridad y no repudio abordados anteriormente.

De esta forma es imprescindible la implementación de un sistema de seguridad que realice las siguientes funciones:

1. Autenticación de usuarios y sistemas.
2. Identificación del interlocutor de cada una de las comunicaciones entre sistemas y servicios.
3. Cifrado de datos digitales sensibles.
4. Firmado digital de datos (documentos, software, etc.) que lo requieran.
5. Asegurar las comunicaciones.
6. Garantía de no repudio (negar que cierta transacción tuvo lugar).

Independientemente existen otros parámetros importantes que deben ser validados para aumentar la seguridad del sistema, como son:

- Seguridad en las contraseñas (Historial y Restricciones).
    - Implementar o crear restricciones en las contraseñas e historiales para lograr una mayor seguridad en los datos, como son el cambio de contraseñas y la eliminación de registros e historiales.
- Chequeos de integridad de los datos. (CheckSum).
    - Para evitar posibles violaciones de seguridad en el servidor de datos se implementa un mecanismo de verificación de integridad de los datos que permita comprobar el estado de los mismos.
- Seguridad dependiente de otros sistemas.
    - Implementar la seguridad del sistema tratando de no delegar funciones de éste en otros sistemas como gestores de bases de datos o servidores web; exceptuando los casos estrictamente necesarios.
- Mecanismo interno de salvas.
    - El sistema tenga implementado un módulo para la realización de salvas automáticas (configurables) o manuales.
- Actualización automática de la bitácora en los siguientes eventos:
    - Administración de usuarios y grupos. Describe cambios de alto nivel a la base de datos de cuentas de usuario, tales como Usuarios creados o cambios en la membrecía de los grupos.
    - Cambios en el Plan de Seguridad. Registra cambios de alto nivel en el Plan de Seguridad, tales como asignación de privilegios.
    - Uso de los derechos de usuario. Intentos exitosos o no de uso de privilegios. Información sobre cuando algún privilegio especial es asignado.

- Sistema: Indica que algo está ocurriendo que afecta la seguridad de todo el sistema o el registro de auditoría.
- Seguimiento de procesos: Suministra información detallada sobre el seguimiento de un proceso.
- Inicio y cierre de sesión: Registra los intentos de inicio y cierre de sesión, tanto exitosa como fallida.
- Accesos a archivos y objetos: Accesos exitosos y fallidos a objetos protegidos

- Seguridad de la comunicación entre los componentes.
  - Para la comunicación entre los componentes arquitectónicos se propone utilizar protección a nivel de transporte. Para ello se debe usar SSL[14] para asegurar la comunicación punto a punto donde el caso lo requiera. SSL proporciona autenticación y privacidad de la información entre extremos de la red por la cual se propaga, mediante el uso de criptografía.

La arquitectura de seguridad debe estar enfocada en 4 dimensiones fundamentales: en el entorno de desarrollo, durante el desarrollo de las aplicaciones, en el despliegue de las aplicaciones y la seguridad de las aplicaciones propiamente.

Seguridad en el entorno de desarrollo

La seguridad en el entorno de desarrollo del software está orientada a garantizar una buena configuración de las herramientas utilizadas en el desarrollo para el soporte de los datos y las aplicaciones, así como la configuración de la herramienta de control de versiones y del lenguaje de programación a utilizar. Aquí se norman estos mecanismos, configuraciones o actividades a tener en cuenta en este sentido.

Guía de actividades para el desarrollo de la Arquitectura de Seguridad

1. **Configuración del servidor Web**

14 Protocolos criptográficos que proporcionan comunicaciones seguras en Internet.

Para evitar ataques y reducir vulnerabilidades en el servidor web se deben realizar las siguientes actividades:

1.1 **Utilizar parches de seguridad:** No tiene sentido poner una cerradura más resistente a tu puerta si dejas la ventana abierta. Del mismo modo si no se tienen los últimos parches de seguridad instalados no tendría sentido continuar con la optimización de seguridad.

1.2 **Desactivar módulos innecesarios:** Actualmente hay gran cantidad de módulos disponibles para servidores web, luego es importante que se conozca el servidor web que se está utilizando para de esta forma saber los que se quieren tener funcionando y los que no.

1.3 **Configurar las cuentas del servidor web:** Se debe crear un grupo para los usuarios que puedan gestionar el servidor, es decir, los que puedan administrarlo, detener e iniciar los servicios necesarios que se llamaran web_admins y otro grupo para los usuarios que van a representar a los servidores Web, que se llamarán web_servers.

Otras de las actividades que se deberían tener en cuenta son:

- Asegurar que los archivos a los que se accede son los deseados.
- Desactivar las opciones para explorar directorios.
- Restringir el acceso por IP*.
- Desactivar los includes* del servidor.
- Disminuir el valor máximo de tiempo de espera.
- Limitar el tamaño máximo de peticiones.
- Ocultar la versión y otra información delicada

2. **Establecer la seguridad en los servidores de las bases de datos.**

En servidores de bases de datos se pueden mencionar 4 niveles básicos de seguridad.

Seguridad de acceso al sistema: Se implementa de dos formas posibles, a nivel de sistema operativo, en cuyo caso el Sistema Gestor de Bases de Datos se apoya en la seguridad de entrada al sistema operativo para comprobar la validez del acceso a los datos almacenados, o bien lo que se llamará modo mixto, en el cual la seguridad de entrada a la información la llevará a cabo el propio servidor de datos a partir de la definición de cuentas de usuarios.

La seguridad a nivel de objetos: Detalla el acceso a nivel de creación y administración de objetos de datos: tablas, vistas, Índices, relaciones, reglas. Es decir las responsabilidades o acciones que el usuario puede hacer en el esquema de base de datos.

La seguridad a nivel de datos: Se realiza en la capa de información, donde se indicará quien puede acceder a que información para su consulta, actualización, inserción o borrado.

Seguridad a nivel de protección de los almacenamientos físicos de la información: Es tarea del sistema operativo, de los archivos de datos del sistema y las políticas de copias de seguridad y restauración de los datos.

Principios o actividades básicas de seguridad en las bases de datos.

2.1 Disgregar responsabilidades: Plantea que el usuario administrador de bases de datos (ABD) debe ser de total confianza y tiene la responsabilidad de mantener la integridad de los datos.
2.2 El ABD y el administrador del sistema operativo (ASO) no deben ser la misma persona.
2.3 Los privilegios del usuario operador y los del ABD no deben ser los mismos.
2.4 Los usuarios no deben compartir sus cuentas ni sus contraseñas.
2.5 Utilizar el principio de mínimo de privilegios*:
2.6 Sólo se debe instalar en los servidores de software requeridos.
2.7 Habilitar sólo los servicios y puertos requeridos.
2.8 La cuenta de administrador sólo la debe tener el personal requerido.
2.9 Restringir las conexiones remotas.
2.10 Aplicar un estándar de seguridad y contraseñas.
2.11 El número mínimo de caracteres de las contraseñas de los usuarios será de 7 caracteres.
2.12 La contraseña del usuario no debe ser igual al nombre del usuario.
2.13 La contraseña debe tener fortaleza requerida con la utilización de letras, números y caracteres especiales.
2.14 La contraseña debe cambiarse como máximo cada tres meses.

3. Mantener un acceso restringido desde la red.

Es recomendable usar siempre un firewall que mantenga protegido al servidor para contrarrestar los ataques. El firewall debe ser configurado para que sólo tenga acceso al servidor, la red o subred requerida, protegiendo con esto los datos.
Se debe configurar además los permisos de acceso a la base de datos, quienes se pueden conectar y a que base de datos, así como los permisos de lectura y escritura sobre tablas y la creación de roles.

4. Configurar la herramienta de control de versiones

Como herramienta para el control de versiones se propone el uso del Subversión.

Subversion es un software de sistema de control de versiones. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como **svn** por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversión es que los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

Además tiene las siguientes ventajas:

- Se sigue la historia de los archivos y directorios a través de copias y renombrados.
- Las modificaciones (incluyendo cambios a varios archivos) son atómicas.
- La creación de ramas y etiquetas es una operación más eficiente. Tiene costo de complejidad constante (O (1)) y no lineal (O(n)) como en CVS*.

Entre otras más.

Como clientes svn se propone el uso de TortoiseSVN para Windows y RapidSVN para Linux.

TortoiseSVN es un cliente Subversion, implementado como una extensión al Shell*. Es software libre liberado bajo la licencia GNU GPL*.

Características:

- Integración con el shell de windows.
- Puede ser usado sin un entorno de desarrollo.
- Pequeñas imágenes decoran los íconos de los archivos mostrando qué archivos o directorios necesitan ser enviados al repositorio.
- Disponible en 28 idiomas diferentes.
- Maneja el mostrar la diferencia de documentos de Office tales como los creados con Microsoft Word.

RapidSVN es un cliente gráfico para Subversion, un programa de control de versiones sustituto de CVS. Es fácil de usar, tanto por quienes ya conocen Subversion como para quien empieza, pudiendo acceder a direcciones SVN, subir y descargar contenido y sincronizarlo con el servidor original, comprobar su estado, crear y fusionar direcciones, etc.

Características:

- Simple: Proporciona una interfaz fácil de usar para las características de Subversion
- Eficiente: Sencillo para los principiantes, pero lo suficientemente flexible como para aumentar la productividad para los usuarios experimentados de Subversion.
- Portátil: Funciona en cualquier plataforma en la que Subversion y wxWidgets se pueden ejecutar: Linux, Windows, Mac OS / X, Solaris, etc.
- Rápido: enteramente escrito en C + +
- Multilingüe: ha sido traducido a muchos idiomas ya: alemán, francés, italiano, portugués, ruso, ucraniano, chino simplificado, japonés.
- Soporte completo para Unicode.

5. Aplicar mecanismos de salva de seguridad

Realizar salvas de seguridad de los datos contenidos en el servidor de control de versiones resulta de gran importancia ya que de esta forma se asegura que toda la información del proyecto incluyendo el código fuente se encuentre resguardada en una copia diferente.

Se deben realizar salvas de la BD todos los días en el servidor destinado para este fin y 3 veces al día en una PC cliente.

Además de éstas se proponen las siguientes actividades:

- Realizar salva del repositorio y de la base de datos en un servidor distinto al servidor de desarrollo dentro del nodo.
- Realizar salva del repositorio y de la base de datos en un disco duro externo.
- Mantener las salvas por un período de 7 días.
- Guardar las versiones estables se en un directorio con acceso restringido.
- Igualmente se propone que el sistema tenga implementado un módulo para la realización de salvas automáticas (configurables) o manuales.
- También se le deben realizar salvas al servidor LDAP*.

Es recomendable que durante la explotación del software se realicen salvas al menos una vez al día.

6. Realizar auditoría de eventos

Las trazas permiten en el desarrollo de software crear un mecanismo de registro oficial de eventos durante un período de tiempo en particular, además de registrar datos o información sobre quién, que, cuando, donde y porque un evento ocurre para un dispositivo en particular o aplicación. Todo esto permite monitorear las actividades de la aplicación o dispositivo, donde se puede obtener una buena oportunidad para determinar eventos y tomar la acción necesaria para corregir el problema o iniciar una investigación en caso de un incidente de seguridad.

La auditoría de las aplicaciones se realiza mediante la utilización de un componente traza. Este componente permitiría el registro de los eventos que ocurren en el sistema permitiendo de esta forma poder corregir problemas que se presenten. Estos eventos serían:

Inicio de una acción: Se dispara un evento al comienzo de una acción.

Terminación de una acción: Se dispara un evento al finalizar una acción.

Error en una acción: Se dispara un evento cuando en una acción ocurre un error.

7. Seguridad durante el desarrollo de las aplicaciones

Otra de las actividades que se realizan en la Vista de Seguridad es establecer la seguridad durante el desarrollo de las aplicaciones, esta se asegura mediante la implantación de una política de codificación segura y un estándar de validación en las diferentes capas de la arquitectura.

- **Política de codificación segura:**

La primera de estas dos actividades, la Política de codificación segura se aplica en las distintas capas como la Capa de Presentación, Capa de negocio y Capa de acceso a datos.

- **Estándares de Validación:**

La segunda de estas actividades, los Estándares de Validación también se establecen en las tres capas mencionadas en la actividad anterior.

8. Seguridad en el despliegue de las aplicaciones

Es necesario garantizar la seguridad durante la explotación de la aplicación para garantizar la integridad y la protección de los datos ante cualquier ataque o contingencia. Para ello se debe realizar una configuración segura tanto en el servidor de aplicaciones como en el servidor de base de datos, además se debe tener implementado algún mecanismo para el control de la integridad de los

datos y del código fuente, y se debe realizar la ofuscación del código para protegerse de ataques de ingeniería inversa.

Entre las actividades que se deben aquí realizar están:

8.1 Configurar de forma optima del servidor web.
8.2 Configurar el servidor de BD.
8.3 Utilizar mecanismos de control de integridad.
8.4 Proteger el código fuente.
8.5 Optimizar y proteger la base de datos.
8.6 Utilizar un ofuscador de código*.

9. Seguridad de las aplicaciones

La seguridad de las aplicaciones se gestionará mediante un Sistema de Gestión Integral de Seguridad. Este sistema tendrá como objetivo fundamental lograr una seguridad centralizada de todas las aplicaciones, lograr la compartimentación de la información, que cada usuario sólo pueda realizar las acciones a las cuales tiene permiso dentro de un sistema y la configuración de su ambiente de trabajo, administrar las conexiones a la base de datos y permisos en ellas, controlar los accesos a los servicios entre sistemas y mostrar un entorno amigable para la realización de todas las acciones deseadas en el sistema.

2.5 Vista de la Arquitectura de Presentación.

La Vista de presentación en la Arquitectura de Software, tiene la responsabilidad como disciplina de definir y desarrollar las tecnologías para la representación de la información y usabilidad de esta. Entre las responsabilidades que cubre se encuentran los aspectos tecnológicos asociados a garantizar atributos de calidad en el producto como son los aspectos de rendimiento, usabilidad, reutilización y portabilidad de la capa de presentación en la arquitectura.

Rendimiento

CSS (Cascade Style Sheets u Hojas de Estilo en Cascada)

CSS es un lenguaje de hojas de estilos creado para controlar la presentación de los documentos electrónicos definidos con HTML*, XHTML, entre otros. Es la mejor forma de separar los contenidos y su presentación y es imprescindible para la creación de páginas web complejas.

Eficiencia de uso de CSS en la WEB.

Con CSS se logra aumentar la densidad de las palabras clave dentro de los contenidos, ya que muchas de las etiquetas ocupan muchísimo menos espacio. Esto también supone un menor peso para las páginas Web. Otras de las características que tributan la eficiencia de uso de CSS es que conserva el ancho de banda del usuario, lo que acelera la carga de páginas, especialmente en conexiones telefónicas. Así como reduce la sobrecarga del servidor y del ancho de banda y reduce el tiempo de diseño y programación.

A la hora de desarrollar una arquitectura de presentación, hay aspectos que por su importancia se deben tener en cuenta a la hora de elegir si se usa o no CSS.

1. **Separación del contenido y presentación.**

Las hojas de estilo generalmente se encuentran en archivos separados del código principal. Esto permite que en un equipo de trabajo, tanto el programador como el diseñador puedan realizar sus tareas de forma independiente aunque paralelas, sin correr el riesgo de que haya interferencias entre ambos, y para ello no alterará el resultado final.

2. **Flexibilidad.**

Las hojas de estilo permiten cambiar en cualquier momento alguna parte o la totalidad del diseño de las páginas con sólo modificar la hoja de estilo, sin que modifique el contenido.

3. Optimización de los tiempos de carga y de tráfico en el servidor.

Al dividir el contenido y apariencia se obtiene archivos más ligeros, y reportándose dos beneficios: por un lado, se reduce notablemente los tiempos de carga del sitio en el navegador. También se une la capacidad de éste para mantener la hoja de estilo en caché.

4. Accesibilidad y estructuración.

La combinación de CSS y marcadores descriptivos posibilita que la página se vea correctamente con o sin hoja de estilos, donde la información se mantiene estructurada y ordenada. Lográndose la accesibilidad sin ningún tipo de problemas tanto por navegadores antiguos o sin soporte para CSS, como para personas con algún tipo discapacidad.

Por todos estos aspectos que regulan en gran medida el rendimiento de la aplicación es que se recomienda el uso de CSS en la implementación de aplicaciones web.

Uso y gestión de imágenes y hosting.

Al igual que cualquier otro tipo de información, las imágenes en un ordenador se encuentran codificadas por un patrón o formato específico. Existen numerosos formatos gráficos para la web, los cuáles se dividen en dos tipos de gráficos, los gráficos de mapa de bits basados en una retícula de píxeles y los gráficos vectoriales basados en información numérica sobre la geometría de los mismos, influyendo estos grandemente tanto en el rendimiento como en la usabilidad y portabilidad.

Tabla 9. Comparación entre uso y rendimiento de diferentes formatos.

JPG	GIF	PNG
• Número de colores: 24 bits color o 8 bits B/N • Muy alto grado de compresión. • Formato de compresión con pérdida. • No permite trasparencia	• hasta 256 colores • Formato de compresión sin pérdida. • Carga progresiva • Permite transparencia • Animación simple	• Color indexado hasta 256 colores y no indexado hasta 48 bits por pixel. • Mayor compresión que el formato GIF (+10%) • Compresión sin pérdida. • Transparencia variable.

• No permite animación.		• No permite animación.
OTROS ASPECTOS		
Ideal para grandes imágenes fotográficas, para impresión y la Web. La calidad disminuye cada vez que se guarda el archivo, por lo que no conviene emplearlo para archivar originales.	Ideal para páginas Web con gráficos pequeños. No es recomendable para colocar fotografías de alta calidad en la Web o para archivar originales.	Produce archivos grandes para la reproducción de fotografías o cuadro comparados con los equivalentes archivos JPG para la Web. No permite crear pequeñas animaciones como el formato GIF.

A partir de la información expuesta en la tabla anterior se estima que el mejor formato para imágenes en la Web es el que se necesite de acuerdo al uso que se le vaya a dar, si se necesitan imágenes de poco tamaño, se puede usar el GIF, si por el contrario se necesitaran imágenes que den efectos de transparencia y buena calidad entonces se recomienda el PNG, o si se necesita imágenes de buena calidad, poco peso sin tener en cuenta la transparencia de capas y que soporte compresión sin tener en cuenta la pérdida de calidad, se usa el JPG. Tener en cuenta todo esto a la hora de elegir imágenes para la Web es muy importante, puede condicionar aspectos como el rendimiento, la usabilidad y la calidad. Se tiene que conocer también las características de la aplicación, de esto depende la correcta elección de las imágenes requeridas, según las necesidades reales de uso.

Se recomienda también tener las imágenes en un servidor distinto al de la aplicación, haciendo que esta solo guarde referencias a las imágenes verdaderas que estarán en el servidor de imágenes, favoreciendo esto la carga de las páginas, ya que dividiendo responsabilidades aumenta el tiempo de respuesta, el tamaño, y mejora considerablemente el espacio en los servidores.

Entre las tecnologías para desarrollar proyectos web se recomienda el uso de Ajax, que es una técnica de desarrollo web para crear aplicaciones interactivas. Estas se ejecutan en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla.

Las cookies son una parte importante en el desarrollo de la capa de presentación, las cookies representan mecanismos que permiten al servidor almacenar en el cliente información relativa a la transacción. Así, cuando el servidor responde a una petición del cliente, inserta una información que

se conservará en el cliente. Esta información es una descripción relativa a un conjunto de direcciones URL. En adelante, será insertada por el cliente en una petición que referencie estas direcciones URL y se denomina una cookie.

Su importancia dentro de la presentación radica en que:

- Es posible realizar la autenticación, efectuar el rastreo, mantener información específica, preferencias y compras de los usuarios, manteniéndose en el disco duro del cliente o bien en la base de datos temporal que crea el sitio.
- Su principal ventaja es que no se requiere que el usuario registre ningún tipo de información, ya que el sitio coge directamente la que aparece en las cookies.

Otro de los elementos a tener en cuenta es **la caché,** que es la encargada de almacenar en un medio físico (HD) los elementos más usados por el sistema (nomencladores, XML y otros) que se almacenan con una estructura que, a pesar de no encontrarse en la memoria volátil del sistema, la recuperación de los mismos es casi instantánea.

El uso de la caché en los navegadores de internet, está tan extendido que incluso el protocolo HTTP* ha incluido ciertas cabeceras para facilitar su control. Cuando se accede por primera vez a un dato, se hace una copia en la cache; los accesos siguientes se realizan a dicha copia, haciendo que el tiempo de acceso medio al dato sea menor

Esto demuestra la importancia de la caché en el sistema y a continuación se expondrán más elementos que demuestran su utilidad, para que los arquitectos de presentación la tenga en cuenta a la hora de implementar una arquitectura de presentación

- Reduce el ancho de banda consumido.
- Disminuye la carga de los servidores.
- Disminuye el retardo en la descarga.

Usabilidad

La Usabilidad es una disciplina que se ha consolidado como una rama de los estudios sobre Interacción Hombre- Máquina, “se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso". (9)

Teniendo en cuenta este concepto se infirieren los siguientes principios básicos al hablar de usabilidad.

- **Facilidad de Aprendizaje**: facilidad con la que nuevos usuarios desarrollan una interacción efectiva con el sistema o producto.
- **Flexibilidad**: relativa a la variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información.
- **Robustez**: es el nivel de apoyo al usuario que facilita el cumplimiento de sus objetivos. Está relacionada con la capacidad de observación del usuario, de recuperación de información y de ajuste de la tarea al usuario.

Es muy importante a la hora de desarrollar la capa de presentación tener en cuenta la usabilidad, en una investigación desarrollado por Sun Microsystem se obtuvieron los siguientes resultados:

- La usabilidad demuestra reducciones del ciclo de desarrollo de los productos de 33-50%. (10)
- El 63% de todos los proyectos de desarrollo de software sobrepasan su presupuesto, siendo las cuatro causas más importantes relacionadas con usabilidad. (11)
- El porcentaje de código que se dedica al desarrollo de la interfaz con los usuarios ha ido aumentando a lo largo de los años hasta un promedio 47-60% del conjunto de la aplicación. (12)

Ya hace algunos años que no se puede hablar de calidad en un proyecto web sin haber tenido en cuenta la usabilidad, hasta el punto que algunos expertos opinen sobre el tema, como el gurú de la usabilidad en los entornos web, Jakob Nielsen, quien definió la usabilidad en el 2003 como "un atributo de calidad que mide lo fáciles de usar que son las interfaces web" o Redish, otro gurú en el área, para quien es preciso diseñar sitios web para que los usuarios sean capaces de "encontrar lo que necesitan, entender lo que encuentran y actuar apropiadamente… dentro del tiempo y esfuerzo que ellos consideran adecuado para esa tarea".

Otro de los conceptos estrechamente asociados al tema de la Usabilidad es **la Accesibilidad** que se refiere a la capacidad de acceso a los contenidos por todas las personas independientemente de la discapacidad (física, intelectual o técnica) que presenten o de las que se deriven del contexto de uso (tecnológico o ambiental). Esta cualidad está íntimamente relacionada con la usabilidad.

Cuando los sitios web están diseñados pensando en la accesibilidad, todos los usuarios pueden acceder en condiciones de igualdad a los contenidos. Por ejemplo, cuando un sitio tiene un código XHTML semánticamente correcto, se proporciona un texto equivalente alternativo a las imágenes y a

los enlaces se les da un nombre significativo, esto permite a los usuarios ciegos utilizar lectores de pantalla o líneas Braille para acceder a los contenidos. Cuando los vídeos disponen de subtítulos, los usuarios con dificultades auditivas podrán entenderlos plenamente, etc.

Es muy importante tener en cuenta la accesibilidad para que nuestra aplicación sea capaz de llegar a todos sin importar la discapacidad que se tenga y se debe hacer un diseño e implementación teniendo esto en cuenta.

Un escenario es una descripción de un personaje que utiliza un producto para conseguir un fin. (13)Los escenarios suelen ser relatos que cuentan una historia en la que se describe una o más tareas desarrolladas en una situación ambiental concreta. Con el desarrollo de escenarios se suelen identificar aspectos importantes que afectan a la utilización de un producto en el mundo real y que no se pueden identificar ni tenerse en cuenta de otro modo. Los escenarios que tienen en consideración la accesibilidad contienen detalles de cómo interactúa con el producto un personaje en condiciones limitantes utilizando estrategias de adaptación y, a menudo, tecnologías de apoyo, su importancia y la necesidad de ellos se ve a la hora de imaginarse como se vería el producto en una situación real, así se pueden identificar los problemas con que interactúa el producto, deficiencias y alcances.

Portabilidad

El Modelo de Objetos de Documento o DOM, es la interfaz que permite acceder y manipular, mediante la programación, los contenidos de una página web (documento HTML o XML.). Proporciona una representación estructurada, orientada a objetos, de los elementos individuales y el contenido de una página, con métodos para recuperar y fijar las propiedades de dichos objetos. Además, proporciona métodos para agregar y eliminar dichos objetos, permitiéndote crear contenido dinámico.

Para desarrollar una arquitectura de presentación se tiene que tener en cuenta algunos elementos del DOM, para tener un mejor desarrollo en la capa de presentación, hay que tener en cuenta la creación y destrucción de objetos, que muchas veces se crean y no se destruyen apropiadamente, dando lugar a errores de sobrecarga de la página y su consecuente caída.

Las versiones 3.0 de los navegadores Internet Explorer y Netscape Navigator introdujeron el concepto de **Browser Object Model (BOM o Modelo de Objetos del Navegador)**, empleado para acceder y modificar las propiedades de las ventanas del propio navegador. Permite a las aplicaciones JavaScript interactuar con las ventanas del navegador. La interacción es independiente del contenido de los documentos HTML.

Entre las mayores ventajas que se pueden encontrar con su uso están:

- Crear, mover, redimensionar y cerrar ventanas de navegador.
- Obtener información sobre el propio navegador.
- Propiedades de la página actual y de la pantalla del usuario.
- Gestión de cookies.
- Objetos ActiveX en Internet Explorer

EL BOM está compuesto por una serie de objetos relacionados entre sí, que son los que le permiten las funcionalidades empleadas en él, aquí se verán cuáles son:

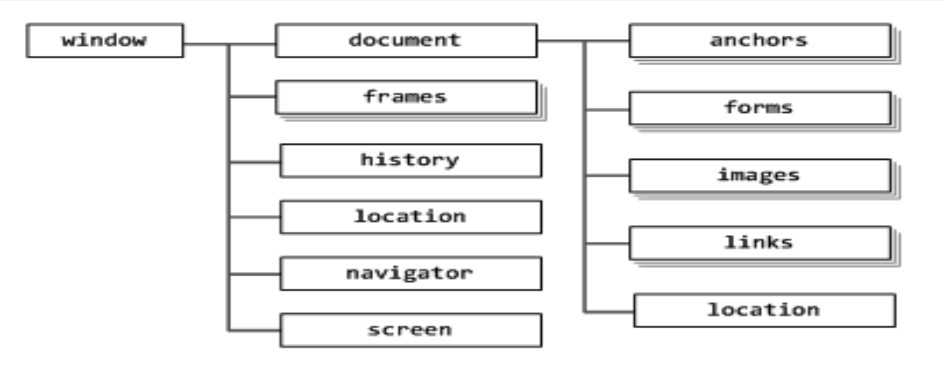


Ilustración 25. Jerarquía de objetos que forman el BOM

Todos estos eventos y objetos se manipulan mediante el objeto window.

El mayor inconveniente del BOM es que su comportamiento no es estándar entre los navegadores, por lo que no es muy apropiado su uso sin antes haber determinado su comportamiento en los diferentes navegadores.

Otro de los aspectos que se tienen que tener en cuenta al definir una arquitectura de presentación es **la Interoperabilidad** que es la capacidad de los sistemas de Tecnologías de la Información y las Comunicaciones (TIC), y de los procesos empresariales a los que apoyan, de intercambiar datos y posibilitar la puesta en común de información y conocimientos.

En pocas palabras, la interoperabilidad es una vía ya probada para resolver la diversidad y heterogeneidad que ofrece el mercado, de esta forma se pueden conectar diferentes software y productos sin afectar su integridad, entre las mayores iniciativas para dotar a la Web de interoperabilidad se encuentran los servicios web.

Los ocho principios que define el Marco Europeo para garantizar una completa Interoperabilidad son:

- Accesibilidad
- Multilingüismo
- Seguridad
- Protección de datos de carácter personal
- Subsidiariedad
- Uso de estándares abiertos
- Valorar los beneficios de software de fuentes abiertas
- Uso de soluciones multilaterales

“Algunas veces se confunde interoperabilidad con software de código abierto. Interoperabilidad se refiere a cómo distintos sistemas de software pueden trabajar juntos”. (Bill Gates)

Reutilización

La Reutilización de Software aparece como una alternativa para desarrollar aplicaciones y sistemas de Software de una manera más eficiente, productiva y rápida. La idea es reutilizar elementos y componentes de Software en lugar de tener que desarrollarlos desde el principio.

Este es un concepto muy importante dentro de la Arquitectura, ya que posibilita reducir grandemente el tiempo de realización de una aplicación, mediante la reutilización de componentes, estándares y normas. Su importancia en la actualidad es crítica, ya que se realiza en un 70% de las aplicaciones o proyectos en la actualidad y es uno de los requisitos indispensables en cualquier proyecto.

Para lograr esto en una medida más eficaz se crearon una serie de patrones que ayudan en el desarrollo de esta tarea. Ellos son:

Patrones de reutilización: El propio nombre de patrón indica que es reutilizable, ya que un patrón es una solución a un problema que puede ser usado muchas veces, hay muchos tipos de patrones que son reutilizables, se tienen los patrones de diseño que se aplican, como su nombre indica, en la fase de diseño, y muchos otros patrones de diferentes tipos aunque generalizados quedarían así:

- **Fundamentales:** Los patrones de esta categoría son los más fundamentales e importantes patrones de diseño conocidos. Estos patrones son utilizados extensivamente en otros patrones de diseño.
- **De Creación:** Los patrones de creación muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas

dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades.

- **De partición:** En la etapa de análisis, se examina el problema para identificar los actores, casos de uso, requerimientos y las relaciones que constituyen el problema. Los patrones de esta categoría proveen la guía sobre cómo dividir actores complejos y casos de uso en múltiples clases.
- **Estructura:** Describen la forma como se pueden relacionar, diferentes tipos de objetos, para trabajar unos con otros y formar estructuras de mayor tamaño.
- **Conducta:** Describen la forma cómo organizar, administrar, y combinar, conductas y responsabilidades de objetos.
- **Concurrencia:** Describen como coordinar operaciones concurrentes para compartir recursos o secuenciar dichas operaciones.

Guía de actividades para el desarrollo de la Arquitectura de Presentación:

1. **Especificar una Nomenclatura para definir componentes:** Cuando se define un componente dentro de una interfaz específica, siempre se cumple un estándar de nomenclatura para nombres, permitiendo de una forma simple que cuando se vaya a analizar un código determinado, se pueda identificar a qué, o de qué tipo de componente se hace referencia.

2. **Validación de Componentes:** La validación es el proceso que permite comprobar la validez de los datos introducidos en función de los requisitos establecidos. De esta manera si se intenta introducir en un campo un valor que no corresponde con la definición, la validación tiene que controlar esta contingencia y mostrar un mensaje u otro tipo de acción similar indicando el no cumplimiento de las precondiciones.

3. **Integración de Interfaces:** Para integrar las interfaces debe utilizarse los los elementos y bondades que pueda implementarse en una interfaz que responde a una lógica determinada, desde otra totalmente independiente que necesite del servicio. Es similar a utilizar un servicio web, pero no a nivel de lógica sino a nivel de interfaces.

4. **Creación de plantillas:** Es una plantilla específica que sigue las normas de estandarización requeridas.

5. **Definir un estándar para el desarrollo de interfaces:** Es la definición de un conjunto de normas para que haya uniformidad en el código y que sea entendible para cualquier programador.
6. **Identificación de componentes:** Es el proceso en el cual se identifican los componentes que formarán parte de la presentación.

2.6 Vista de la Arquitectura de Tecnología.

La Arquitectura de Software se divide en diferentes áreas, donde cada una de ellas tiene sus roles y responsabilidades. Esta vista tiene como objetivo especificar y describir las tareas y competencias de los roles así como las actividades y artefactos del área de tecnología, y por tanto, definir las habilidades y los conocimientos que deben desarrollar las personas que trabajan en esta área de la Arquitectura de Software para lograr un buen desempeño.

El Área de Tecnología es la responsable de garantizar un soporte tecnológico para el desarrollo de las configuraciones que propone el resultado de una arquitectura de sistema, así como las bases tecnológicas para los frameworks especializados de la Arquitectura de Sistema o de otras áreas como la de integración. Esta es el área responsable de identificar las tecnologías y herramientas a usar en la realización de la aplicación, además de definir la factibilidad técnica del producto.

En esta disciplina se define la mejor arquitectura posible utilizando una tecnología en particular, igualmente es aquí donde se dan las soluciones técnicas para la optimización de la aplicación, es decir para garantizar la portabilidad, flexibilidad y rendimiento de la aplicación. Los integrantes del Área de Tecnología son los responsables de generar una tecnología tipo para la aplicación que se desea desarrollar, de transmitir el conocimiento al resto del proyecto y por último de implantar y controlar el uso correcto de la tecnología, ganando autoridad y prestigio dentro del proyecto.

Las responsabilidades del área de tecnología son las actividades que realiza el Arquitecto Tecnológico y los artefactos que genera. Este capítulo se dividirá en dos partes, en la primera parte se describirán las actividades que realiza el área y en la segunda los artefactos que genera.

Guía de actividades para el desarrollo de la Arquitectura de Tecnología:

1. Valoración y evaluación de los escenarios tecnológicos de acuerdo al tipo de aplicación que se desea desarrollar y selección del escenario más óptimo según las características propias del proyecto, los atributos de calidad de la aplicación y los recursos materiales y financieros disponibles.

2. Valoración y evaluación de las plataformas tecnológicas para el escenario tecnológico seleccionado y selección de la plataforma más óptima teniendo en cuenta las características propias del proyecto, los atributos de calidad de la aplicación, los recursos materiales y financieros y el nivel de conocimiento de los recursos humanos disponibles en estas plataformas.

3. Valoración y evaluación de los frameworks y librerías para la plataforma tecnológica seleccionada y selección de los frameworks y librerías más óptimas según las características propias del proyecto, los atributos de calidad y las categorías o aspectos arquitectónicamente significativos (cache, transacciones, excepciones, etc.) de la aplicación, los recursos materiales y financieros y el nivel de conocimiento de los recursos humanos disponibles.

4. Descripción y especificación del estilo arquitectónico, los frameworks y librerías, los componentes con sus conectores, relaciones y restricciones de acuerdo a los atributos de calidad y las categorías o aspectos arquitectónicamente significativos de la aplicación.

5. Desarrollo de frameworks, librerías y/o componentes verticales arquitectónicamente significativos que no existan ó extensión de los seleccionados que no satisfagan del todo los atributos calidad y/o den solución a las categorías o aspectos arquitectónicamente significativos de la aplicación.

6. Desarrollo de un marco de trabajo tipo para la aplicación que se desea desarrollar, integrando los frameworks, librerías y componentes desarrolladas, extendidas y/o seleccionadas.

7. Elaboración de las pruebas de concepto de las tecnologías seleccionadas, desarrolladas o extendidas para de esta forma determinar sus restricciones arquitectónicas.

8. Elaboración de los documentos rectores y de control necesarios para el aseguramiento de la calidad de la aplicación desde el punto de vista tecnológico, dígase estilos de codificación, listas de chequeo para controlar el uso correcto de la tecnología, guías tecnológicas, prácticas

y patrones, roles y responsabilidades de los involucrados en el uso de la tecnología, métricas, etc.

9. Elaboración de los documentos de apoyo y ayuda a los involucrados en el uso de la tecnología, manuales de usuario, manuales de instalación y configuración, casos de estudios, etc.

10. Descripción y especificación técnica del marco de trabajo tipo de la aplicación que se desea desarrollar, elaborando la documentación técnica del mismo.

11. Elaboración de cursos de capacitación para capacitar a los involucrados en el uso del marco de trabajo tipo de la aplicación que se desea desarrollar.

12. Elaboración del expediente legal del marco de trabajo de la aplicación que se desea desarrollar, que incluye las licencias de los frameworks, la licencia propia del marco de trabajo y el dictamen legal.

Estos artefactos que se generan en esta Vista Tecnológica son:

- Expediente tecnológico.
  - Especificación tecnológica del marco de trabajo.
    - Documento de la vista tecnológica.
    - Documentos de especificación técnica de los componentes.
  - Especificación legal del marco de trabajo.
    - Licencias de las tecnologías usadas.
    - Licencia del marco de trabajo.
    - Documento del dictamen legal del marco de trabajo.
  - Manuales del marco de trabajo
    - Manual de usuario
    - Manual de instalación
    - Documentos de los casos de estudio.
  - Paquetes estables del marco de trabajo
    - Documento de notas y control de versiones.
    - Paquetes estables del marco de trabajo.
    - Casos de estudio.
- Gestión de la calidad.

- Documento de estilo de código.
- Documentos de listas de chequeo.
- Documentos de prácticas y patrones.
- Documentos guías.
  - Guía de arquitectura tecnológica.
  - Guía del marco de trabajo.
- Documento de métricas.
- Documento de roles y responsabilidades.

- Preparación y superación
  - Curso de arquitectos de tecnología.
  - Curso de programadores técnicos.

Conclusiones del Capítulo

En este capítulo se expuso un breve estado del arte acerca de la actualidad en el tema de las disciplinas de la Arquitectura de Software para visualizar y justificar el por qué de las propuestas que se establecieron en este trabajo. Además se definieron 6 vistas principales, Sistema, Seguridad, Presentación, Integración, Datos y Tecnológica. Exponiéndose las actividades necesarias para desarrollar cada una de estas vistas, los artefactos que se generan en cada una y además las definiciones principales para entenderlas.

Capítulo 3. Responsabilidades del Arquitecto de Software.

Introducción del Capítulo

Son muchas las opiniones alrededor de cuáles son los roles que deberían conformar un equipo de arquitectura, qué es lo que debería hacer cada uno y cómo se integrarían entre ellos. Las principales compañías que se dedican a la producción de software han brindado sus propuestas pero enfocadas a las necesidades particulares que tienen. Así, en este capítulo, se presentarán algunas de las clasificaciones de esas compañías, y al final la propuesta de los autores de este libro, donde están clasificados los roles agrupándolos dentro de dos dimensiones. Las actividades que realiza cada uno y los conocimientos que son necesarios para realizarlas, son los otros dos temas que se tratarán en el presente capítulo.

3.1 Rol del arquitecto

Nacimiento del término

El nacimiento de los arquitectos se remonta al inicio de la vida del hombre en la tierra aunque no existiera la palabra como tal y es que para hablar de un Arquitecto se tiene que necesariamente hacer referencia a su significado etimológico. Esta palabra proviene de los griegos, quienes bautizaron dicho papel con la palabra ἀρχιτέκτων (architécton), definiendo al director de una construcción. Esta palabra proviene de la unión de dos raíces muy fuertes ἀρχός (archós), que significa “guía” y τέκτων (técton) que significa “constructor”. A nuestro idioma llegó gracias a los romanos que llamaron Architectus a los grandes guías de las impresionantes y avanzadas obras civiles del imperio más grande del mundo antiguo.

Es polémico el tema a la hora de definir como tal qué es un Arquitecto de Software, esto puede ser debido a que ubican al arquitecto en el contexto de una organización en particular ajustándolo a sus necesidades. Pero sí se puede a grandes rasgos explicar las principales responsabilidades y competencias que debe tener asociado este rol.

Las palabras tienen su propio significado. Un arquitecto es un arquitecto, no un ingeniero, no un programador, no un científico, ni un webmaster* o un director de proyecto. La palabra arquitecto es distinta en el negocio de la construcción. En la construcción de software, muchos se apropian de la importancia del título, pero fallan en representar bien el rol. El arquitecto construye, no desarrolla. Los

edificios no son desarrollados. Desarrollar es hacer crecer, evolucionar y descubrir. Los arquitectos ven en perspectiva la construcción, no guían el desarrollo.

Definiciones del Rol del Arquitecto de Software

*1.El Rol del Arquitecto de Software según Peter Eeles de IBM**

El Arquitecto "es una persona, equipo u organización responsable por la arquitectura del sistema" (IEEE 1471).

- ✓ Es un líder técnico. El arquitecto tiene competencias técnicas y de liderazgo, debe tener la autoridad para tomar decisiones técnicas, ayuda a armar el equipo y a organizar el trabajo, además constantemente comunica el valor de lo que se está haciendo.
- ✓ El rol puede ser llenado por un equipo con un líder claro. No siempre una persona tiene todas las competencias. Un "Equipo es un pequeño grupo de gente con competencias complementarias y comprometidos con el propósito, y con enfoques comunes por los que son mutuamente responsables".
- ✓ El arquitecto entiende el proceso de desarrollo de software. El arquitecto debería tener conocimiento del proceso de desarrollo, ya que éste garantiza que todos los miembros del equipo trabajen de manera coordinada. Un buen proceso define las funciones claramente. Dado que el arquitecto participa a diario con muchos de los miembros del equipo, es importante para el arquitecto entender sus funciones y responsabilidades. A diario los desarrolladores apoyan su trabajo en el arquitecto preguntando cómo hacer tal cosa, por lo tanto, existe una clara coincidencia entre el papel del arquitecto y el papel de gestor del proyecto.
- ✓ Entiende el dominio del negocio. Un dominio es un "área de conocimiento o actividad caracterizada por un conjunto de conceptos y terminología entendida por los profesionales del área" y permite imaginar requisitos "probables" y anticipar cambios.
- ✓ Tiene conocimiento tecnológico, pero no necesariamente experticia profunda. Dado que la tecnología cambia con cierta frecuencia, es esencial que el arquitecto se mantenga actualizado con los cambios tecnológicos.
- ✓ Tiene competencias de diseño.
- ✓ Tiene suficiente competencia de desarrollo como para comunicarse con el equipo.
- ✓ Es un buen comunicador.
- ✓ Toma decisiones.
- ✓ Entiende la "política" de la empresa.

- ✓ Es un negociador, debe explicar a stakeholders del proyecto las consecuencias de sus opciones o alternativas de arquitectura. (14)

2.El Rol del Arquitecto de Software según el Software Engineering Institute (SEI)

¿Cuáles son las responsabilidades, competencias y conocimientos de un Arquitecto de Software?
Para responder a esta pregunta, desde hace unos cuantos años el SEI (Software Engineering Institute) ha estado recopilando comentarios de las personas sobre las responsabilidades de un Arquitecto de Software. Este es un compendio, en el que se resaltan los aspectos más comúnmente citados (todas las respuestas individuales y sus autores pueden ser consultadas en el sitio web del SEI) y que a continuación se enumeran:

- ✓ Elaborar la arquitectura correcta para solucionar el problema o necesidades del cliente.
- ✓ Definir y documentar la solución elaborada. Asegurarse que todos los involucrados la estén utilizando y la estén utilizando bien.
- ✓ Asegurarse que se aplica en etapas de manera coordinada de tal forma que toda la organización pueda apropiarse de ella antes de que sea completada.
- ✓ Asegurarse de que la Arquitectura de Software sea acorde con el sistema deseado.
- ✓ Actuar como el embajador de la propuesta arquitectónica.
- ✓ Hacer que la gerencia la entienda hasta el nivel necesario.
- ✓ Asegurarse de que el modelamiento sea correctamente realizado.
- ✓ Conocer cuáles cualidades sistémicas, como el rendimiento, deben alcanzarse y en qué medida.
- ✓ Responder sobre las inquietudes relacionadas con la selección de herramientas y ambientes de desarrollo.
- ✓ Identificar e interactuar con los interesados en el proyecto para asegurarse que sus necesidades son satisfechas.
- ✓ Asegurarse que la arquitectura no es solamente la correcta para la operación del sistema, sino que además es la correcta para su soporte y evolución.
- ✓ Resolver conflictos y ayudar a generar acuerdos.
- ✓ Solucionar problemas de tipo técnico.
- ✓ Mantener la moral, tanto en el interior del grupo de arquitectura como al exterior. Esto último es realizado proponiendo un diseño compacto cuando se requiera y entregando presentaciones y materiales que le permitan a todas las personas en la organización saber que se encuentran en el camino correcto.
- ✓ Entender y planear las rutas de evolución del sistema, diseñar un plan que guíe la adopción de nueva tecnología.

- ✓ Gerenciar las estrategias de identificación y mitigación de los riesgos asociados con la arquitectura.

3. El Rol del Arquitecto de Software según Rational Unifed Process (RUP)

El Arquitecto de Software tiene la responsabilidad general de conducir las principales decisiones técnicas, expresadas como la Arquitectura de Software. Por lo general, esto incluye la identificación y documentación en la arquitectura de los aspectos importantes del sistema, incluidos los requisitos, diseño, implementación y despliegue, es decir, las vistas del sistema.

El arquitecto se encarga también de proporcionar la justificación de estas decisiones, buscar el equilibrio entre los stakeholders participantes, haciendo disminuir los riesgos técnicos, y garantizando que las decisiones sean comunicadas, validadas y adoptadas efectivamente.

El Arquitecto de Software debe poseer la madurez, visión y la experiencia que le permita comprender los problemas de manera rápida y tener un juicio crítico cuando existe información incompleta, por ejemplo. Más concretamente, el Arquitecto de Software, o miembros del equipo de arquitectura, deben combinar estas capacidades:

- ✓ ***Experiencia*** en el dominio del problema, a través de una profunda comprensión de los requisitos, y el dominio de la Ingeniería de Software. Si hay un equipo, estas cualidades se pueden propagar a través de los miembros del equipo, pero al menos un Arquitecto de Software debe tener la visión global del proyecto.
- ✓ ***Liderazgo*** con el fin de gestionar el esfuerzo técnico a través de los diversos equipos, y tomar las decisiones acordes aún bajo presión. Para ser eficaz, el Arquitecto de Software y el jefe de proyecto deben trabajar en estrecha colaboración. El Arquitecto de Software debe tener la autoridad para tomar decisiones técnicas.
- ✓ ***Comunicación*** para ganar confianza, para persuadir, motivar, y guiar. El Arquitecto de Software no puede conducir por decreto, sólo con el consentimiento del resto del equipo del proyecto. El Arquitecto de Software debe ganarse el respeto del equipo y del director del proyecto, del cliente, la comunidad de usuarios, así como el equipo de gestión.
- ✓ ***Orientación por objetivos y pro-actividad*** con un implacable enfoque en los resultados. El Arquitecto de Software es la fuerza impulsora detrás del proyecto, no un soñador o visionario. La carrera de un exitoso Arquitecto de Software es una larga serie de “óptimas” decisiones adoptadas en la incertidumbre y bajo presión. Sólo los que pueden centrarse en hacer lo que hay que hacer tendrán éxito en este entorno del proyecto.

Desde el punto de vista de expertos, el Arquitecto de Software (Arquitectura de Software) también debe abarcar el Papel de Diseñador, sin embargo, a diferencia del diseñador, el Arquitecto de Software:

- ✓ Tiende a ser un generalista en lugar de un especialista, a sabiendas de muchas tecnologías de alto nivel en lugar del nivel de detalle.
- ✓ Hace más amplias las decisiones técnicas en el dominio de la solución, y por lo tanto, debe tener amplio conocimiento y experiencia, así como la comunicación y habilidades de liderazgo son fundamentales.

El Arquitecto de Software es un rol en un proyecto de desarrollo de software, en el cual se realizan varias actividades y se tienen responsabilidades como se muestra en el siguiente gráfico.

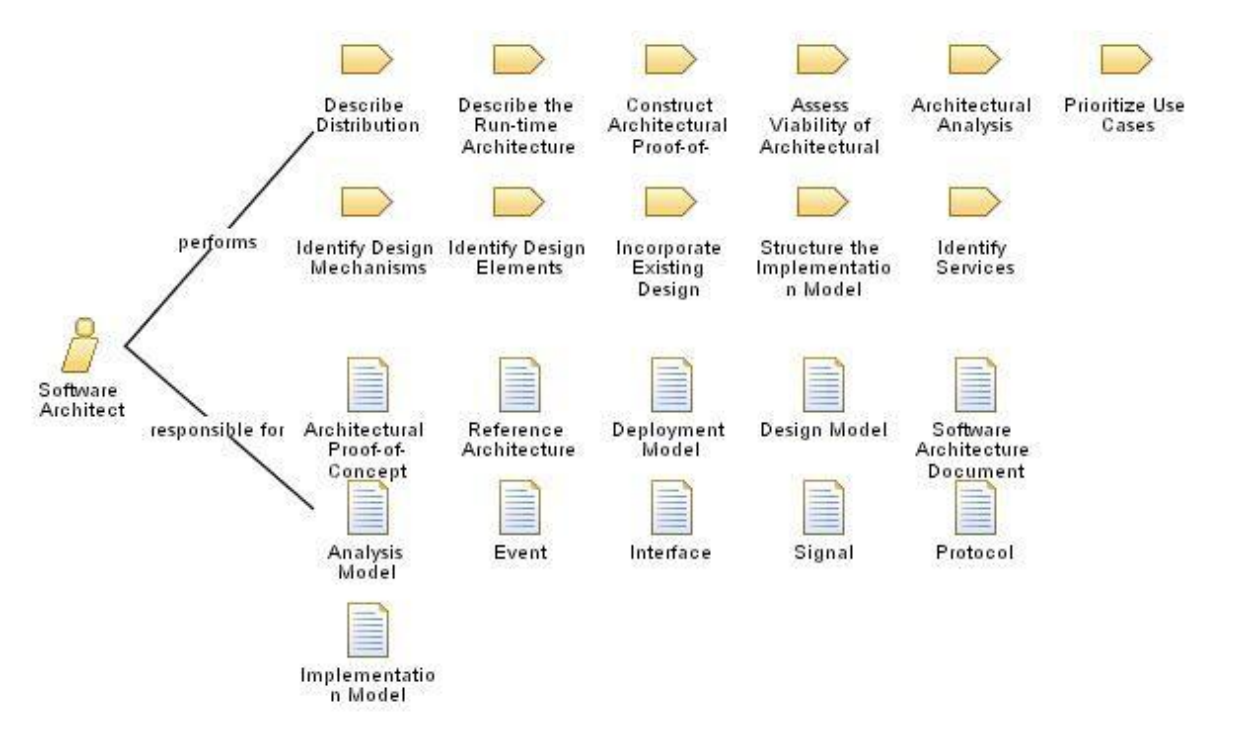


Ilustración 26. El Rol del Arquitecto de Software según (RUP)

4.El Rol del Arquitecto de Software según Microsoft

Microsoft clasifica los arquitectos de la siguiente forma:

- ✓ ***Enterprise Architect/Chief Architect***: El Arquitecto Empresarial es el responsable de la ejecución de la visión del CIO y la estrategia de TI. Incluye la definición de programas estratégicos, la selección de plataformas tecnológicas adecuadas, y proporcionar orientación para las implementaciones. El Arquitecto Empresarial ayuda al CIO a asegurar que las inversiones en TI estén alineadas a la estrategia de negocio, y a proporcionar ventaja

competitiva para la organización. Es también quien define las normas, directrices y los lineamientos de gestión, para adaptar su aplicación a las normas y directrices definidas. En algunas organizaciones, esta tarea se fusiona con la del CIO.

- ✓ ***Solution Architect***: El arquitecto de Soluciones es el responsable de la ejecución de un programa estratégico de TI. Esto incluye la definición de la solución arquitectónica para el programa, la selección de plataformas tecnológicas acordes a la estrategia de la empresa, comunicación con el equipo de trabajo, y la toma de decisiones sobre cuestiones técnicas durante la ejecución del proyecto. Generalmente tiene que mediar entre las empresas y equipos de tecnología y otros grupos. En algunas organizaciones, este papel se define simplemente como "arquitecto". El puesto de alto nivel tiene el título de "Arquitecto Líder".
- ✓ ***Technical Architect***: El arquitecto técnico es por lo general un especialista en una tecnología particular. Esta persona tiene conocimiento experto de la tecnología y las funciones de la misma, los componentes que la integran, y comprende sus limitaciones y puntos fuertes. Es responsable de determinar la aplicabilidad de la tecnología, para definir la mejor arquitectura posible utilizando una tecnología en particular, y también para guiar al equipo en la aplicación de la solución. En general, del arquitecto técnico se espera conocer las distintas herramientas de proveedores en el ámbito de la tecnología, las últimas tendencias en el mercado, de arquitectura y diversas alternativas para aplicar la solución.

La siguientes gráfica muestra la relación entre estos tres roles con la tecnología y la estrategia de la organización.

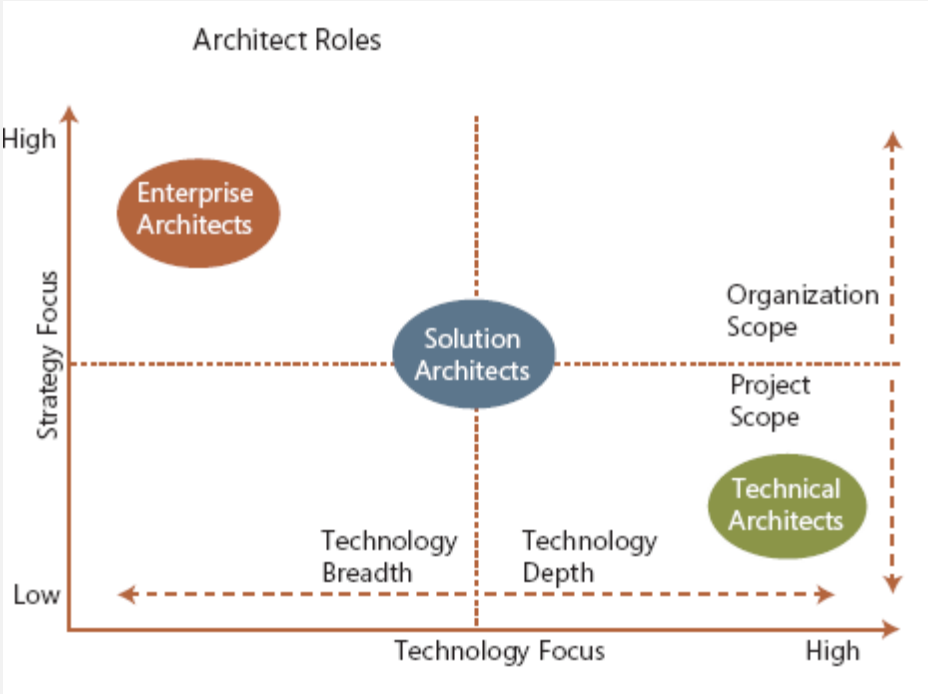


Ilustración 27. Relación Entre Roles

Además existen los arquitectos denominados:

- ✓ ***Infrastructure architects*.** El Arquitecto de Infraestructura es responsable de las decisiones del área de infraestructura, de mantener el entorno de TI y los usuarios finales, y de comunicarse constantemente con los ingenieros que mantienen áreas específicas de la infraestructura. Se encargan de crear una arquitectura que cumple con los acuerdos de niveles de servicio de las necesidades de los empresarios y apoya las aplicaciones y soluciones que se requieren para operar en el día a día de las empresas.

Microsoft en su programa de "Arquitecto Microsoft" considera algunas características comunes a todos los arquitectos independientemente del tipo de arquitecto. Algunas de estas características son:

- ✓ ***Poseer fuerte visión para los negocios*:** Consiste en entender los costos de capital operacional y considerar cada uno de estos mientras se crea la solución. Leer estados financieros, tener conversaciones con funcionarios financieros y tener una comunicación acertada con los dueños de negocios para justificar los proyectos y calcular el rendimiento de un proyecto.
- ✓ ***Pensamiento visionario:*** Durante la participación en un proyecto, el arquitecto debe considerar y proyectar la tecnología en el futuro, visionando los cambios que se producen en los negocios de los clientes, y la mejor manera de aprovechar las ventajas de la solución tecnológica actual en el futuro.
- ✓ ***Investigar nuevas tecnologías***: El arquitecto debe estar en continua investigación de nuevas tendencias en tecnología, arquitectura de TI y las aplicaciones empresariales.
- ✓ ***Comprender Frameworks arquitectónicos y las mejores prácticas***: Los arquitectos entienden cuáles son los Frameworks de arquitectura y empresariales y su valor en un proyecto. Los arquitectos seleccionan y usan metodologías en los proyectos, entienden el funcionamiento de Frameworks y cómo la solución será desarrollada, y el comportamiento antes y después del despliegue. Entienden el ciclo de vida de un proyecto y de una solución.
- ✓ ***Seguir y divergir a la vez***: Cuando se trabaja en un entorno particular o en un proyecto específico, los arquitectos deben tener la capacidad de personalizar o modificar Frameworks y/o las metodologías utilizadas para lograr una solución a un problema o requisito de negocio.
- ✓ ***Poder para desarrollar rápidamente profundo conocimiento en una tecnología*:** Ganando profundidad en múltiples tecnologías anteriores, el arquitecto puede asociar o transferir la capacidad de aprender otros métodos para investigar y ganar rápidamente experiencia en nuevas tecnologías.

- ✓ ***Pueden trabajar con información ambigua o incompleta:*** Los Arquitectos deben colaborar en el proceso de indagación de la información para llegar a una solución, pero pueden empezar a trabajar con información limitada y conforme el proyecto progresa, tomar decisiones de compensación o equilibrio con el fin de mantener una solución que cumpla con los objetivos, y continuar satisfaciendo las exigencias de negocio que al principio fueron identificadas. Sin embargo el arquitecto debe saber claramente si con la información limitada puede empezar a trabajar sin poner en riesgo el proyecto más adelante por cambios drásticos o si el proyecto debe suspenderse antes de recopilar información mínima para empezar las tareas, es importante el trabajo conjunto de todo el equipo de proyecto en este aspecto.

Microsoft posee un programa de certificación de Arquitectos (Microsoft Certified Architect Program), el cual sirve para identificar a los mayores expertos en Arquitectura TI del sector. Se trata de arquitectos que pueden utilizar múltiples tecnologías para resolver problemas empresariales y ofrecer cifras y parámetros a los negocios para ayudarles a determinar el éxito o el fracaso de los proyectos que dirigen.

Competencias de un arquitecto según Microsoft.

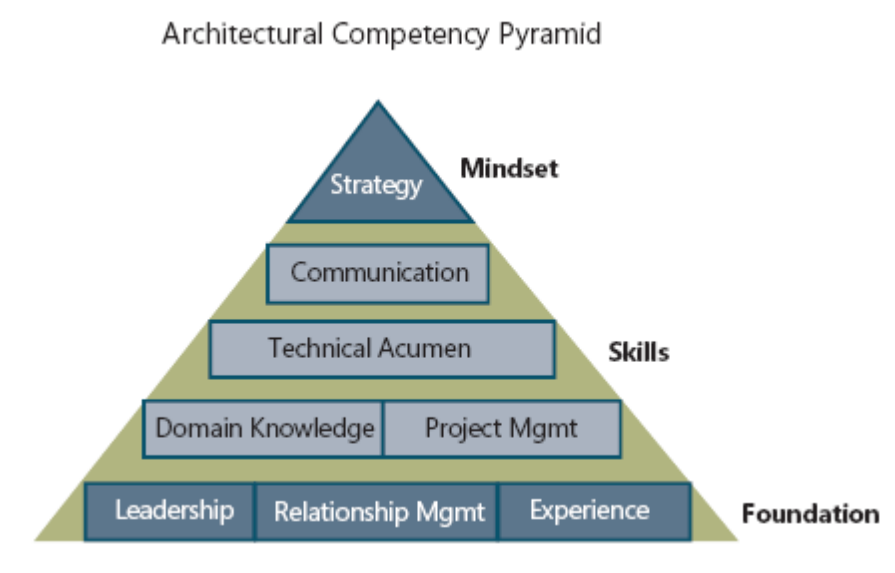


Ilustración 28. Competencias del Arquitecto de Software

En esta pirámide, la experiencia y cualidades de liderazgo constituyen los pilares fundamentales del rol del arquitecto. También se necesita perspicacia técnica, buenas habilidades de comunicación, entender el dominio del problema antes de diseñar una solución y la capacidad de gestión.

El Arquitecto de Software debe tener una mentalidad estratégica, es decir, la habilidad de ver las cosas a 50.000 pies de altura, a un nivel estratégico, abstraerse de la complejidad operativa. Se trata de adoptar una visión más amplia.

Muchos recursos educativos y las certificaciones están disponibles para alcanzarlas. Además los arquitectos con experiencia, son otra fuente importante de recursos, ya que la información por sí sola es insuficiente para el desarrollo de muchas habilidades necesarias. Los aspirantes a arquitectos deben considerar muchos factores a la hora de hacer carrera, como los tipos de proyectos para el acceso a los mentores o expertos. La arquitectura no solo es una exigente pero gratificante profesión, sino que requiere determinación y una buena planificación para desarrollar plenamente las habilidades del arquitecto y madurar en el papel.

5.El Rol del Arquitecto de Software según Bredeeyer Consulting

Puede definirse de manera simple que un Arquitecto es aquél que hace Arquitecturas y sus responsabilidades se restringen a hacer bien su trabajo. Esto puede incluir articular la visión arquitectónica con las necesidades del cliente, conceptualizar y experimentar con diferentes estrategias arquitectónicas; crear modelos, componentes y documentos de especificación de interfaces.
Sin embargo, cualquier arquitecto experimentado sabe que el papel no sólo encierra estas tareas de tipo técnico, sino que existen otras de un carácter más diplomático y estratégico por un lado y por otro lado tareas de consultoría y asesoría. Un sentido coherente del negocio y una adecuada estrategia tecnológica son necesarios para vislumbrar la arquitectura que solucionará los problemas del cliente, dados los objetivos y restricciones al arquitecto en la organización. Las actividades en esta área incluyen la escucha activa a los interesados del proyecto para entender de manera profunda sus intereses y las metas a satisfacer, implica también crear mapas tecnológicos y estrategias de diferenciación, a la par con la realización de afirmaciones sobre tendencias tecnológicas y sus consecuencias en la estrategia técnica del proyecto y la arquitectura planteada.

El arquitecto (o equipo de arquitectura) necesita tener empatía con una variedad de grupos de interesados en el proyecto, incluyendo la gerencia a diferentes niveles, analistas de negocio o de

ventas y sobre todo los desarrolladores. El arquitecto necesita balancear su participación con la necesidad de tomar en cuenta las múltiples opiniones de su equipo de trabajo. Mientras más amplio horizonte tenga la arquitectura, más ajustada será a la óptima. El arquitecto tiene que pasar por encima de muchas "Políticas Organizacionales" para lograr convencer a muchos interesados en el proyecto, para comunicar extensivamente y trabajar con diversas redes de personas que influyen en el éxito de la arquitectura.

Pero lograr "vender" la arquitectura no es suficiente. Todos los que estén vinculados con su implementación necesitan entenderla. Los documentos tipo "ladrillo" son famosos por ser excelentes "recogedores de polvo". La participación temprana de los desarrolladores más experimentados trae buenas ideas en el proceso de definición de la arquitectura y también crea un amplio entendimiento de los deseos de los desarrolladores y el costo de su implementación.
Adicionalmente para proyectos grandes, puede ser bastante útil crear tutoriales que expliquen cuáles fueron las decisiones que llevaron a proponer la arquitectura objetivo, ya que, durante los diversos ciclos de construcción es necesario realizar ajustes.

El arquitecto también es un mentor y un entrenador, trabajando con los desarrolladores para motivarlos cuando los retos aparecen, sobre todo cuando tienen un amplio impacto o son críticos para el éxito del sistema. Más allá, el arquitecto no sólo debe liderar su equipo y la comunidad de desarrolladores sino que debe liderar y motivar la organización completa en la dirección técnica adecuada.

¿Quién es adecuado entonces para el papel de Arquitecto?, pues, muy frecuentemente el convertirse en "Arquitecto" es una promoción ofrecida a los desarrolladores muy hábiles en un esfuerzo por retenerlos. Desafortunadamente no todos los técnicos superdotados tienen el talento y las competencias que los hacen buenos arquitectos. Aún así, el título genera expectativas, en el "arquitecto" y en el resto de la organización, de las responsabilidades asociadas con el cargo. Esto puede generar un montón de conflictos para una persona fuertemente orientada a la parte técnica, que repentinamente se ve enfrentada con políticas empresariales y exigencias de una comunicación efectiva y fluida. Los mejores arquitectos entonces, son expertos en tecnología que infunden respeto en la comunidad técnica, también son buenos estrategas, excelente diplomáticos, consultores, asesores y líderes.

Para desarrollar en 1995 un taller en Hewlett-Packard sobre arquitectura, Bredemeyer Consulting estudió muchos proyectos arquitectónicos, revisó literatura al respecto e incluso estudió el proceso de la Arquitectura de edificios. Tal sería el inicio de una larga relación con cientos de arquitectos de

diferentes campos de la industria que le han permitido a esta empresa consultora un entendimiento profundo de la arquitectura y del proceso arquitectónico como tal.

Basados en este entendimiento, y mirando las tendencias actuales de la arquitectura, se ha podido establecer un Marco de Trabajo donde se han identificado varias áreas de actividad críticas o dominios de competencia, que aparecen definidamente en el papel de arquitecto. Estos son Tecnología, Estrategia Organizacional, Política Empresarial, Asesoría y Liderazgo. Cada uno de ellos tiene sus propios elementos de conocimiento, actividades y características personales que hacen de un arquitecto ser exitoso en cada uno de dichos aspectos. Estos elementos pueden ser agrupados según apunten a las competencias del Saber, del Saber Hacer y del Saber Ser. El siguiente cuadro muestra cómo se relacionan entre si estos elementos. (15)

Tabla 10. Relación entre Tecnología, Estrategia Organizacional, Asesoría y Liderazgo.

	What you KNOW	What You DO	What You ARE
Technology	In-depth understanding of the domain and pertinent technologies Understand what technical issues are key to success Development methods and modeling techniques	Modeling Tradeoff analysis Prototype/experiment/simulate Prepare architectural documents and presentations Technology trend analysis/roadmaps Take a system viewpoint	Creative Investigative Practical/pragmatic Insightful Tolerant of ambiguity, willing to back-track, seek multiple solutions Good at working at an abstract level
Consulting	Elicitation techniques Consulting frameworks	Build "trusted advisor" relationships Understand what the developers want and need from the architecture Help developers see the value of the architecture and understand how to use it successfully Mentor junior architects	Committed to others' success Empathetic, approachable An effective change agent, process savvy A good mentor, teacher
Strategy	Your organization's business strategy and rationale Your competition (products strategies and processes) Your company's business practices	Influence business strategy Translate business strategy into technical vision and strategy Understand customer and market trends Capture customer, organizational and business requirements on the architecture	Visionary Entrepreneurial
Organizational Politics	Who the key players are in the organization What they want, both business and personal	Communicate, communicate, communicate! Listen, network, influence Sell the vision, keep the vision alive Take and retake the pulse of all critical influencers of the architecture project	Able to see from and sell to multiple viewpoints Confident and articulate Ambitious and driven Patient and not Resilient Sensitive to where the power is and how it flows in your organization
Leadership	Yourself	Set team context (vision) Make decisions (stick) Build teams Motivate	You and others see you as a leader Charismatic and credible You believe it can and should be done, and that you can lead the effort You are commited, dedicated, passiorate You see the entire effort in a broader business and personal context

Analizando las definiciones anteriores y otras experiencias, los autores de este trabajo definen al Arquitecto de Software como la persona encargada de asesorar en la planificación, dirigir, seguir y controlar todas las tareas técnicas relacionadas con el desarrollo del software, y que juega un papel significativo a la hora de esclarecer los objetivos que a menudo son inciertos o contradictorios, no es un soñador ni un visionario, es el director técnico de un proyecto. Al Arquitecto de Software lo caracteriza el poseer una gran habilidad para trabajar consistentemente en un nivel abstracto. Tiene la responsabilidad de dirigir las principales decisiones técnicas, expresadas como la Arquitectura de Software. Por lo general, esto incluye la identificación y documentación de la arquitectura de los

aspectos importantes del sistema, incluidos los requisitos, diseño, implementación y despliegue, es decir, las vistas del sistema.

El arquitecto también es responsable de proporcionar el fundamento de estas decisiones, equilibrando las preocupaciones de los diferentes interesados, reduciendo los riesgos técnicos, garantizando que las decisiones se comuniquen, se validen con eficacia y se acaten.

Roles en la Arquitectura de Software

En Epidata Consulting, la primera empresa especializada en Arquitectura de Software de América Latina, se engloban todas las disciplinas en solamente 3 bloques:

Arquitecto Técnico

El arquitecto técnico es por lo general un especialista en una tecnología particular, tiene un conocimiento experto de la tecnología y de las funciones de la misma, los componentes que la integran, y comprende sus limitaciones y puntos fuertes. Esta persona es responsable de determinar la aplicabilidad de la tecnología, para definir la mejor arquitectura posible utilizando una tecnología en particular, y también para guiar al equipo en la aplicación de la solución. En general, del Arquitecto Técnico se espera conocer las distintas herramientas de proveedores en el ámbito de la tecnología, las últimas tendencias en el mercado, de arquitectura y diversas alternativas para aplicar la solución.

Arquitecto Funcional

Tienden a ocupar el rol de líder de equipo. Maneja el proyecto y planea junto al jefe de proyecto las iteraciones. Suele representar un canal de comunicación fluida entre el jefe de proyecto y los equipos de desarrollo. Valida diseños; guía a los desarrolladores para que cumplan con las expectativas de calidad tomando métricas, organizando y promoviendo la documentación y las buenas prácticas; conformando las vistas de la arquitectura o de sistema y asegura que el proyecto no se desvíe de la arquitectura previamente definida.

Arquitecto Corporativo

Unifica los dos casos mencionados anteriormente; pero con algunos agregados.

IBM, Microsoft, SEI, SUN, RUP y Bredemeyer* concuerdan con los perfiles definidos anteriormente y solo agregan una categoría más:

Arquitecto de Infraestructura

Este rol se encarga más de planificar el hardware, la red, el software básico como el sistema operativo, la base de datos, y también herramientas de monitoreo que ayuden a controlar que todo eso funcione sin problemas. A este arquitecto no le concierne cómo las aplicaciones fueron desarrolladas o lo que las mismas hacen -en términos de negocio- sino que las ve más como procesos que deben estar ejecutándose en forma "saludable".

Muchas empresas ven al arquitecto como un rol innecesario, como un perfeccionismo, que solo demora el desarrollo y que su labor es prescindible. Pero todo esto es justamente por el desconocimiento que hay sobre el tema, y más aún por la gran variedad de definiciones y responsabilidades diferentes que se le asocian.

3.2 Dimensiones del rol del Arquitecto de Software.

Al Arquitecto de Software se debe ver desde dos dimensiones, una corporativa y otra de desarrollo. ¿Por qué verlo así? Se parte del entorno de negocio y se llega a las partes, el arquitecto en esta dimensión inicialmente se enfoca en la visión del software, ve el producto como el objetivo a alcanzar, analiza la forma en que se desarrolla, define cómo y con qué se integra, cuáles son hitos tecnológicos y de negocio, define sus diferentes configuraciones, proyecta una imagen de cómo quedará el producto y qué se necesitará para construirlo. En esta dimensión se ve el producto desde la industria o la organización, se piensa como empresa, se analizan las oportunidades de negocio, se visualizan las integraciones estratégicas necesarias, y esto es lo que se corresponde con la dimensión corporativa. O sea una visión desde el negocio, sin desarrollar nada, totalmente de trazar estrategias, de valoración de oportunidades, de evaluación del producto y de alcance.

Ya en un segundo momento, se necesita llevar a cabo todo eso que se planeó en la primera dimensión, lo que se proyectó y que era hasta el momento una idea, ahora es cuando tiene que hacerse realidad, entonces se está en presencia de la dimensión de desarrollo. Aquí se definen y

estructuran componentes, conectores, artefactos, diseño, datos, componentes. Se fundamenta el interior del producto y cada una de sus partes.

En la primera dimensión se encontrará un Arquitecto Corporativo como líder al frente de un equipo de trabajo que cuenta con un Arquitecto de procesos que tendrá a su disposición un equipo de trabajo formado por analistas y funcionales. Este arquitecto se encargará de definir los procesos necesarios, es el visionario, el que evalúa las oportunidades de negocio, que tan factible será construir el software, que tiempo se necesita, etc. Este equipo lo integra también un Arquitecto de Sistema que evalúa las dimensiones del producto, propone las soluciones, y por último un Arquitecto Tecnológico que analiza las necesidades tecnológicas para llevar a cabo el desarrollo del software. Este arquitecto tendrá a su cargo un equipo de tres roles: Tecnológico, Seguridad y Presentación que se encargarán de visualizar las necesidades del software en cada uno de los temas que ellos tratan.

En la dimensión de desarrollo se mantiene el Arquitecto de Procesos continuando con las responsabilidades anteriormente mencionadas, surge el Arquitecto Temático (uno por cada línea[15] que tenga el proyecto), que será como el de sistema, pero se encargará de una única línea temática y que estará al frente de un arquitecto de datos específicamente de esa línea temática, están además el Arquitecto de Integración al frente de todos los temas de integración y un Arquitecto de Datos al frente de todos los temas referente a la BD. Todos estos arquitectos estarán subordinados al Arquitecto de Sistema.
Y dentro de un último grupo bajo la guía del Arquitecto de Tecnología están un Arquitecto de Seguridad, un Arquitecto de Presentación y uno de Tecnología. Además en esta dimensión se creará un grupo de trabajo encargado de aplicar las pruebas de concepto.

[15] Entiéndase en este contexto como Módulo, Subsistemas. Son las partes en que se dividirá un sistema para su implementación. Por ejemplo en el proyecto ERP-Cuba, las líneas son Contabilidad, Costo y Proceso, Capital Humano, Arquitectura…

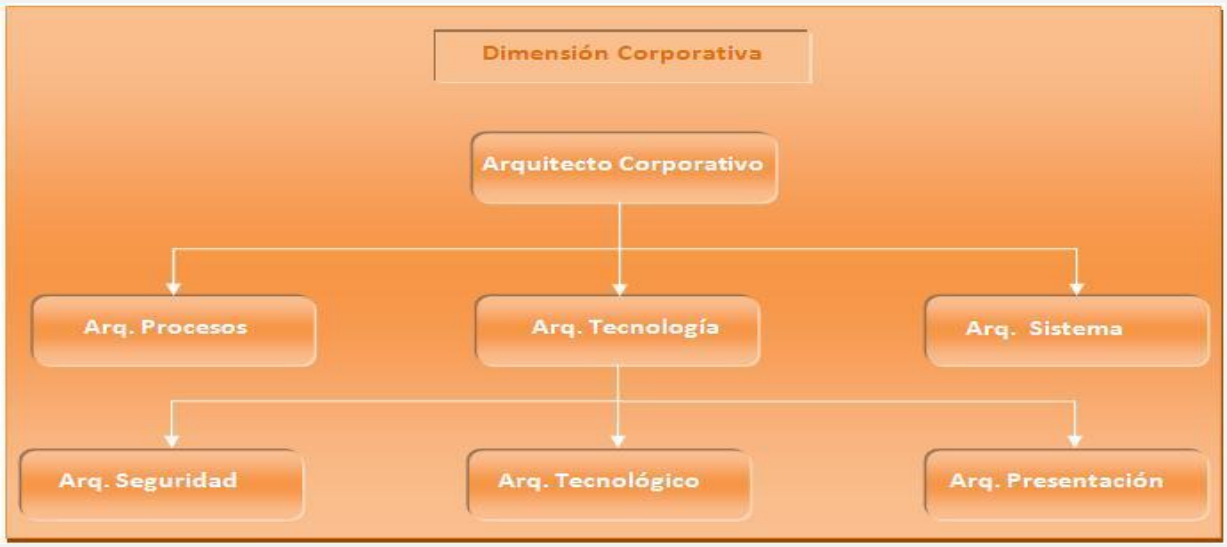


Ilustración 29. Dimensión Corporativa

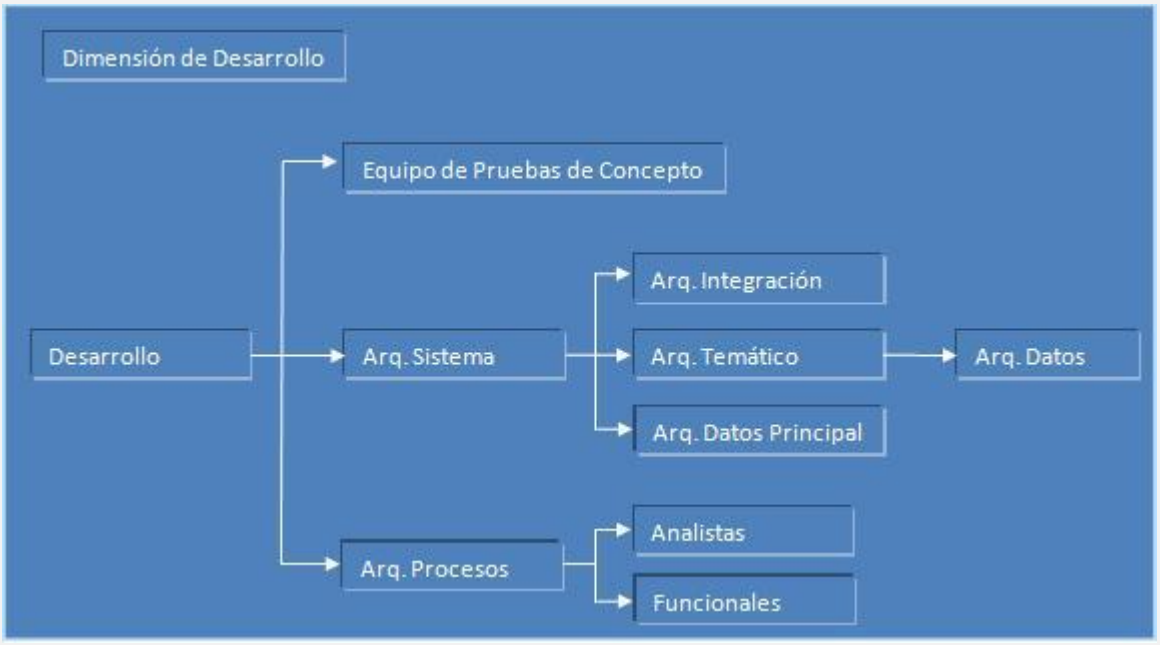


Ilustración 30. Dimensión de Desarrollo

Competencias asociadas al Arquitecto de Software.

El Arquitecto de Software debe reunir una serie de cualidades que al final terminan definiendo la forma de ser y actuar, aunque más que cualidades o características se podrían ver como requisitos indispensables. Debe ser visionario, ser capaz de percibir las oportunidades que se le presenten, tomar decisiones críticas en el momento necesario, estar comprometido con su equipo, superarse constantemente en todas las áreas que le competen, saber dirigirse a los demás con un correcto uso del lenguaje, no redundar a la hora de explicar algún tópico. Por su función como puente entre varios integrantes del proyecto tiene que ser comunicativo. Es imprescindible que posea conocimiento de las tecnologías y que posea un alto nivel de abstracción. Que sepa organizar y liderar un equipo de trabajo. Debe influenciar en la motivación y formación del equipo de trabajo así como tener carisma y jovialidad.
Se entiende como competencias del rol del arquitecto la habilidad de traducir la visión del negocio en visión técnica, llevar las estrategias de negocio a estrategias arquitectónicas y ejecutar eficazmente estas estrategias. Para esto debe dominar las siguientes competencias o temáticas.

- *Tecnología*

El arquitecto debe tener un sólido conocimiento de la razón de ser de la organización, de su infraestructura tecnológica y del proceso de desarrollo de sistemas de información. Sin embargo, a pesar de pertenecer también al ámbito tecnológico las actividades del arquitecto de sistemas difieren de las que comúnmente realizan los desarrolladores. Más allá de tener claridad acerca de los requerimientos específicos del sistema, el arquitecto debe preocuparse por las implicaciones de la selección de una solución tecnológica en los objetivos de la organización, para lo cual debe tener la visión general del sistema y construir los modelos necesarios para representar el problema y su mejor solución, explorando diferentes alternativas, preparando documentos y explicando la arquitectura a los involucrados en el proyecto.

- *Estrategia de negocios*

El arquitecto debe poseer un alto conocimiento de la estrategia y la lógica detrás de los negocios de la organización, así como de los procesos operativos de las diferentes áreas del negocio, los ciclos de planeación y los procesos de toma de decisiones; en resumen, un buen entendimiento del contexto del negocio de la organización.

- *Política organizacional*

Las arquitecturas comúnmente están dirigidas a atender las necesidades de varios y diversos usuarios, por lo que usualmente requieren de la participación de diferentes desarrolladores. Es decir, que serán utilizadas a través de las diferentes áreas de la organización y por diferentes equipos de desarrollo que pueden ser internos o externos. En este sentido el arquitecto necesita entender las expectativas de la organización y de los usuarios con respecto a la arquitectura seleccionada, y debe asegurar que permanezcan comprometidos durante el desarrollo del proyecto.

- *Consultoría*

Durante la construcción de una arquitectura participan diferentes proveedores, desarrolladores y usuarios, por lo cual el papel del arquitecto como consultor es fundamental al asistir a los involucrados del proyecto, que se convierten en sus clientes, en la resolución de dudas, en la preparación de presentaciones y en la coordinación de las actividades necesarias para desarrollar el proyecto con éxito.

- *Liderazgo*

En este sentido el arquitecto actúa como el líder que infunde al equipo una visión común del proyecto, y que lo motiva a trabajar comprometido para alcanzar los objetivos propuestos con la implementación de la arquitectura seleccionada.

El Arquitecto de Software “Posee competencias técnicas y conocimiento tecnológico, investiga nuevas tecnologías y comprende framework arquitectónicos y las mejores prácticas, desarrolla rápidamente un profundo conocimiento en una tecnología. Tiene liderazgo y autoridad, sigue y dirige a la vez, es buen comunicador, entiende el dominio del negocio, es un negociador, tiene fuerte visión para los negocios, entiende la política de la empresa, puede trabajar con información ambigua o incompleta, se identifica e interactúa con los interesados en el proyecto para asegurarse que sus necesidades sean satisfechas. Además se orienta por objetivos y pro-actividad y debe poseer madurez, visión y tener un juicio crítico.” (16)

3.3 Roles del arquitecto en la dimensión corporativa.

3.3.1 Arquitecto Corporativo o Principal:

Es el encargado de asesorar en la planificación y estimación del proyecto, conocer a la perfección los requerimientos, restricciones, necesidades y entorno así como identificar las tareas de implementación, necesidades tecnológicas, hitos tecnológicos y línea base. Debe tener un amplio conocimiento del entorno de desarrollo y es el máximo responsable de las decisiones arquitectónicas.

Definen los estándares, taxonomía, políticas de reusabilidad, patrones arquitectónicos, de diseño y la visión de la organización. Hace cumplir los principios taxonómicos. Lleva a cabo el tratamiento de excepciones, persistencia, transacciones, auditoria y seguridad. Es manager continuo de la integración, proporciona la descripción y especificación arquitectónica, proporciona consejos técnicos y guía al administrador del proyecto. Es quien establece la disciplina técnica del desarrollo, la integración y vista de sistema del producto. Debe asesorar al equipo de gerencia, participar en las discusiones y decisiones arquitectónicas tanto de sistema como tecnológico, que puedan comprometer o tengan alto impacto en la arquitectura del proyecto, y por ende en su desarrollo. Decide cómo tratar el concepto de reutilización, los servicios tecnológicos al desarrollo, construcción tecnológica del producto, etc. Conoce y ayuda a la implementación de los procesos de calidad y define la evaluación y factibilidad técnica de productos, componentes y frameworks.

3.3.2 Arquitecto de Procesos:

Responsabilidades

Este arquitecto aparece tanto en esta dimensión como en la Dimensión de Desarrollo. La diferencia está en el papel que desempeña dentro de cada una. En esta dimensión, las responsabilidades que tiene el Arquitecto de Procesos son desde el punto de vista de gestión. Estas obligaciones y/o tareas son:

- Participar en la planificación del modelado del negocio.
- Dirigir los procesos de modelado de negocio en la línea.
- Integrar los procesos de negocio y modelos conceptuales.
- Revisar los modelos de procesos y de negocio y por último.
- Aplicar métricas para evaluar los procesos de modelado de negocio.

Competencias del Arquitecto de Procesos

Debe poseer una serie de conocimientos o competencias como dominar el tema de la Gestión de alcance y tiempo. Conocer los procesos de modelado de negocio. Saber cómo aplicar métricas para la evaluación de procesos y dominar técnicas de dirección.

Las habilidades con que debe contar este arquitecto para llevar a cabo sus tareas son: saber identificar tareas de modelado de negocio, dependencias entre tareas, estimar duración de las mismas y asignación de recursos, identificar puntos de integración en los modelos de negocio y modelos conceptuales. También debe saber evaluar los modelos de procesos, modelos de negocio y especificaciones de requisitos desarrollados en la línea y saber aplicar métricas para evaluar los procesos de modelado de negocio.

3.3.3 Arquitecto de Sistema

Las organizaciones cada vez son más conscientes de la necesidad de contar con el mejor diseño para operar sus negocios de una manera eficiente y competitiva, pero también perciben el alto costo que esto puede representar. Ante esta situación ha surgido la necesidad de crear un rol conocido como Arquitecto de Sistemas en el cual se ha delegado la responsabilidad de investigar, evaluar y seleccionar las mejores alternativas para atender las necesidades específicas del negocio a un costo razonable.

Un ingeniero de software que acepte asumir el rol de arquitecto de sistemas debe poseer un talento y conocimiento especiales para llevar a cabo el diseño, construcción e implementación de una Arquitectura de Software.

Responsabilidades

Las responsabilidades del arquitecto de sistemas podrían sintetizarse en particular en la visión arquitectónica; conceptualizando y experimentando con diferentes alternativas tecnológicas; creando modelos, componentes y documentos de especificación de interfaces y validando la arquitectura contra los requerimientos y presunciones del impacto de la alternativa seleccionada sobre la estrategia tecnológica de la organización. A continuación se desglosan aún más estas responsabilidades, siendo el arquitecto de sistemas el responsable de:

Identificar componentes: Actividad más compleja que realiza el arquitecto de sistema. Esta se obtiene del análisis de los requisitos y casos de uso definidos en el negocio, también de los modelos conceptuales que conforman los analistas. Es compleja además porque no hay un estándar o patrón a seguir para la identificación de los componentes, de ahí que el arquitecto de sistema deba realizar un cuidadoso y metódico diseño apoyándose a su vez en estilos y patrones que le ayuden a obtener soluciones óptimas a los problemas.

Establecer conectores: Estos serán los contratos mediante los que se realizará la integración entre componentes. Suelen ser servicios que brindará y/o que solicitará cada componente.

Establecer interfaces claras: Las interfaces serán las clases contenedoras de los servicios que se traten, son clases comunes como siempre se ha visto, con sus atributos y métodos (servicios) de ahí la importancia de que se definan con claridad para que cuando se vaya a usar un servicio sea entendible lo que éste brinda y como lo brinda.

Identificar soluciones del sistema: Las soluciones del sistema estarán recogidas en un documento, estas soluciones permitirán crear un producto configurable y que pueda ser aplicado a cualquier sistema contable.

Identificar nodos críticos de integración: Los nodos críticos de integración son los nodos que mayor número de servicios tiene desde y hacia él.

Definir variables globales: Las variables globales son datos de mucho uso en la aplicación que no constituyen componentes, pero que son básicos para la integración. Casi siempre se dividen en variables de sesión y variables de aplicación.

Definir línea Base: La arquitectura que se presenta define la línea que deben seguir los procesos de análisis, diseño e implementación, evaluando las mejores prácticas y estilos vigentes en la actualidad referentes a la Arquitectura de Software que permita a los desarrolladores y demás involucrados tener una idea clara de lo que se está implementando

Definir estándares y normas: Este documento constituye una guía para el desarrollo en las líneas de producción, desde el punto de vista arquitectónico, con el propósito de lograr una estandarización del código.

Priorización de Componentes: la Priorización de Componentes se hace midiendo la cantidad de dependencias que tiene este desde y hacia él, permite además ordenar el desarrollo.

Competencias del Arquitecto de Sistema.

Para cumplir con las responsabilidades señaladas el arquitecto de sistemas debe desarrollar ciertas competencias que le permitan asumir su rol de una manera exitosa. Estas competencias se pueden categorizar en los siguientes dominios: tecnología, estrategia de negocios, política organizacional, consultoría y liderazgo.
Como puntos imprescindibles debe contar con conocimientos sobre estilos arquitectónicos, patrones de diseño, metodologías de desarrollo, etc.

La complejidad de los sistemas de tecnología de información y la interrelación entre sus componentes puede ser mejor comprendida y modelada si es representada como si se tratase de una estructura arquitectónica; y como tal, es el arquitecto de sistemas quién a través del desarrollo de competencias que combinan sus conocimientos tecnológicos con habilidades gerenciales, está capacitado para asistir a la organización en la selección de la arquitectura más adecuada para responder a las necesidades reales del negocio.

3.3.4 Arquitecto de Tecnología

El Área de Tecnología por las responsabilidades que tiene dentro de la arquitectura de un sistema tiene como principal rol al arquitecto tecnológico. Este rol tiene sus tareas y competencias que están orientadas a la función que realizan dentro del área y que permiten definir las habilidades que debe tener cualquier integrante para asumir con responsabilidad y conocimiento este rol en particular.

Responsabilidades

El arquitecto tecnológico cuenta con responsabilidades como la de seleccionar los escenarios arquitectónicos (distribuidos, no distribuidos, rígidos, etc.), la plataforma tecnológica (.Net, J2EE, Linux-PHP-Apache-Postgres, etc.) y los frameworks y librerías para la plataforma tecnológica (para .Net .Net Framework, para PHP Zend Framework, para Java Spring Framework, etc.). Además debe describir y especificar el estilo arquitectónico, los frameworks y librerías, los componentes con sus conectores, relaciones y restricciones. Elabora las pruebas de conceptos de las tecnologías seleccionas, desarrolladas o extendidas para de esta forma determinar sus restricciones

arquitectónicas. Formaliza los cursos de capacitación para capacitar a los involucrados en el uso del marco de trabajo tipo de la aplicación que se desea desarrollar, el expediente legal del marco de trabajo tipo de la aplicación que se desea desarrollar, que incluye las licencias de los frameworks y librerías, la licencia propia del marco de trabajo y el dictamen legal. Debe controlar y asegurar el uso correcto del marco de trabajo tipo de la aplicación que se desea desarrollar y describir y especificar técnicamente el marco de trabajo tipo de la aplicación que se desea desarrollar. Estudia y determina la factibilidad técnica de la aplicación. Controla y gestiona estrictamente los cambios realizados sobre el marco de trabajo y Elabora los documentos rectores y de control necesarios para el aseguramiento de la calidad de la aplicación desde el punto de vista tecnológico.

Competencias del Arquitecto de Tecnología

El arquitecto tecnológico dirige un equipo de arquitectura desde el punto de vista tecnológico, por lo que debe tener un amplio conocimiento de las tecnologías más usadas para el área de negocio de la aplicación que se desea desarrollar, generando tecnologías tipo para esta área, haciendo una vigilancia tecnológica constante en busca de tecnologías más robustas y productivas; debe tener un amplio conocimiento de las prácticas y patrones de diseño y arquitectura, así como de principios de programación; debe ser capaz de asimilar y dominar cualquier lenguaje o paradigma de programación, debe comprender totalmente cualquier framework, siendo capaz de extender el mismo si es necesario.

Este rol debe ser capaz de generar una tecnología tipo, que concentre la programación de la aplicación en el proceso y la lógica del negocio y en las interfaces de usuario, alejando a los programadores de los detalles arquitectónicos, como, tratamiento de excepciones, administración de conexiones y transacciones, validaciones, entre otras categorías arquitectónicamente significativas, debe dar cumplimiento a los atributos de calidad de la aplicación y a las características propias de la misma, cumpliendo con todas las decisiones arquitectónicas que tengan impacto en la tecnología; además responde por el uso correcto de las herramientas y tecnologías usadas durante el proceso de desarrollo, dictando un conjunto de políticas y estándares tecnológicos para guiarlo, aplicando mecanismos de control manuales o automáticos para su estricto cumplimiento.

Los arquitectos tecnológicos son programadores técnicos avanzados que además de tener las habilidades mencionadas anteriormente debe tener características de liderazgo y gestión, con poder y autoridad, deben ser capaces de convencer al resto del equipo de arquitectura que la tecnología

usada es la más óptima para el desarrollo de la aplicación, además debe apoyarse en los programadores técnicos para el cumplimiento de sus tareas dirigiéndolos y orientándolos.

3.3.5 Arquitecto de Seguridad

El arquitecto de seguridad es la persona encargada de velar por la seguridad del sistema en general, desde políticas de acceso a seguridad en la Base de Datos.

Responsabilidades

El arquitecto de seguridad es el encargado de elaborar toda la política de seguridad a utilizar en el entorno de desarrollo y despliegue del software basándose en los principios básicos de seguridad para cualquier sistema informático: Autenticación, Autorización, Administración de perfiles, Administración de conexiones y Auditoria. Además identifica el desarrollo o utilización de componentes que garanticen la protección de la información que se manipula. Define la vista de la Arquitectura de Seguridad para que sirva como guía en el desarrollo e implantación del sistema. Identifica y define el mecanismo de llave pública y llave privada. Define un estándar de validación de los datos y una política de codificación segura. Establece el grado de fortaleza de las claves de los usuarios del sistema y de la base de datos. Conforma una lista de chequeo con todos los aspectos que se deben cumplir para garantizar la seguridad tanto en el entorno de desarrollo como en el de despliegue. Realiza el plan de seguridad informática del centro de desarrollo contemplando políticas y medidas a seguir en el ambiente de despliegue y establece el grado de fortaleza de las claves de los usuarios del sistema y de la base de datos.
Otra de sus responsabilidades es establecer políticas de salvas de seguridad para evitar pérdida de información y garantizar que siempre se cuente con una copia fiable y actualizada.

Todo esto de manera general, ahora acotando las responsabilidades a un nivel más específico:

En el cliente:

- Definir un estándar de validación de los datos para evitar la entrada de datos erróneos o maliciosos y política de codificación segura en el cliente, teniendo en cuenta el manejo seguro de variables globales, acceso a ficheros, etc.

En el servidor web:

- Definir un estándar de validación de los datos y política de codificación segura en el servidor, teniendo en cuenta el manejo seguro de sesiones, cookies, ciclos no abiertos, variables globales, uso de memoria, acceso a ficheros, uso de la cache, paso por parámetros, entre otros
- Identificar los puntos críticos en la configuración del servidor de aplicaciones para buscar las herramientas y métodos a utilizar para la configuración del mismo.

En el servidor de datos:

- Definir la política de configuración a seguir para el servidor de datos.
- Definir la política de acceso a los datos ya sea desde las aplicaciones o directamente en el servidor de datos, para esto debe apoyarse en la creación de roles y usuarios de base de datos, restringiendo los permisos de cada uno.

En la comunicación:

- Definir el mecanismo de llave pública y llave privada.
- Garantizar la seguridad en la comunicación entre el cliente, el servidor web y el servidor de datos asegurando la integridad de los datos apoyándose en los protocolos seguros de comunicación y en los mecanismos de encriptación.

En el ambiente de desarrollo:

- Establecer políticas de salvas de seguridad para evitar pérdida de información y garantizar que siempre se cuente con una copia fiable y actualizada.

Ambiente de despliegue

- Implementar un Mecanismo de CRC* para comprobar la integridad de las aplicaciones web.
- Implementar un Mecanismo de CRC para comprobar la integridad de la estructura de la base de datos.
- Garantizar la protección del código mediante un ofuscador de código.

Competencias del Arquitecto de Seguridad

Para asumir la responsabilidad del arquitecto de seguridad se debe contar con ciertas competencias que faciliten desempeñar este rol. Un arquitecto de este tipo debe poseer amplios conocimientos en el campo de la seguridad informática, tener claridad acerca las 5 A de la seguridad informática (autenticación, autorización, auditoría, administración de conexiones y administración de perfiles), además de ser un visionario ante cualquier ataque, amenaza, riesgo, vulnerabilidad y/o contingencia del que pueda ser objeto la información, pudiendo ser informático o no el ataque. Éste debe tener

sólidos conocimiento acerca de las técnicas de aseguramiento de sistemas como son técnicas de codificación de la información: Criptografía, contraseñas difíciles de averiguar a partir de datos personales del individuo, protocolos de comunicación segura, mecanismos de control de integridad de los datos y las aplicaciones y además conocer acerca de tecnologías protectoras como: cortafuegos, sistema de detección de intrusos - antispyware, antivirus, llaves para protección de software, etc. Un arquitecto de seguridad debe tener cualidades de liderazgo que le permita orientar al equipo de desarrollo del software en este sentido, al mismo tiempo necesita tener conocimientos de las tecnologías que se emplean en la realización del proyecto, en este sentido debe ser capaz de garantizar una configuración segura del servidor de aplicaciones y del de base de datos, debe conocer el lenguaje de programación y las herramientas de control de versiones que se utilizan en el desarrollo del software así certificar su uso seguro en el manejo de información. Debe tener una visión amplia de los posibles entornos donde se desplegará el sistema para crear planes de seguridad informática y también en este sentido debe tener conocimiento de técnicas para la protección del código de las aplicaciones.

3.3.6 Arquitecto de Presentación

Es la persona encargada de llevar a cabo y verificar el proceso de diseño del sitio y que trabaja estrechamente con los diseñadores gráficos y los responsables del mismo para definirlo. Está integrado en un equipo y sus tareas abarcan desde la fundamentación del proyecto hasta el rediseño, verificación y testado del producto durante todas las fases de desarrollo hasta la obtención del resultado final.

Responsabilidades.

Desarrollar y verificar procesos de producción o diseño de información con el fin de que el usuario pueda recuperar la información de un determinado espacio de manera clara, precisa y sin ambigüedades, en cualquier plataforma o soporte; en especial se habla de soportes multimedia e interactivos, aunque no se debe omitir ningún soporte por plano que este sea y hablar de experiencias de usuario.

Organizar, estructurar, sistematizar, rotular, distribuir, diseñar estructuralmente sistemas de información (17)con el fin de que el usuario pueda hacer de su experiencia de recuperación algo simple, agradable, eficaz y productivo.

También durante el desarrollo se encarga de definir:

- El diseño de la interacción.
- El diseño de la navegación y esquemas de facetas.
- El etiquetado o rotulado de los contenidos para acceder a la información.
- Decisiones de reutilización.
- Normas y estándares del desarrollo.
- La facilidad de búsqueda y el diseño de la interfaz de búsqueda.
- Decisiones usabilidad y taxonomía del diseño.
- Nomenclatura para definir componentes.
- Validación de Componentes.
- Integración de Interfaces.
- Estructura de Plantillas.
- Multilenguaje.

Competencias del Arquitecto de Presentacion

Un arquitecto de información debe reunir un mínimo de conocimientos procedentes de diferentes disciplinas, por tanto se hace necesario separar estos conocimientos requeridos en dependencia del área en la que trabaje:

- Diseño gráfico: No implica ser diseñador gráfico, ni dominar por completo una herramienta de diseño. Se refiere a la habilidad de establecer relaciones entre los elementos visuales y determinar su total integración dentro del Web.
- Estar actualizado con las principales prácticas de diseño existentes, estar relacionado con estas tecnologías y tener un conocimiento abarcador de estas.
- Documentación e información: la documentación se basa en el estudio y creación de medios de acceso a la información, así como determinar la forma más apropiada de organizarla para garantizar su posterior recuperación. Estos son métodos adecuados para iniciar una arquitectura de información

- Marketing: los conocimientos sobre investigaciones de interfaces de usuarios o apariencias, así como la identificación de segmentos atractivos del mercado constituyen la labor diaria de estos especialistas. El Web como producto no puede permanecer ajeno a ello.
- Informática: resulta de suma importancia el conocimiento del entorno tecnológico del Web. A partir de ello, pueden establecerse limitantes y definir el alcance de las prestaciones que se desean implementar en el sitio.
- Ingeniería en usabilidad: comprende la habilidad y los métodos para evaluar el funcionamiento del sistema, desde la curva de aprendizaje hasta los errores más frecuentes que cometen los usuarios.

3.4 Roles del arquitecto en la dimensión de desarrollo.

3.4.1 Arquitecto de Procesos

Responsabilidades

Este arquitecto se encarga de interactuar con los clientes para tener una mejor visión de lo que se quiere y de evaluar las oportunidades de negocio. Otras de sus responsabilidades son identificar los objetivos y procesos del negocio, describir estos últimos y organizarlos taxonómicamente. También debe identificar las entradas y salidas de los procesos, elaborar el Mapa de Procesos del Negocio, elaborar el modelo conceptual, identificar las entidades conceptuales dentro de este modelo y describirlas. Además es el responsable de determinar el tamaño y la complejidad de los procesos de negocio.

Competencias del Arquitecto de Procesos

Para que este arquitecto pueda realizar todas sus responsabilidades tiene que poseer dominio de las herramientas CASE, del trabajo con el lenguaje de modelado UML. Debe estar al tanto de conceptos como Procesos de Negocio, Objetivos de Negocio, Entradas y Salidas de los procesos de negocio, Clientes, Mapas de Procesos del Negocio, etc. Siendo un poco más exquisitos, para que desempeñe

aún mejor su trabajo este arquitecto además del trabajo en UML debe saber BPMN y Métricas para el tamaño y la complejidad de los procesos de negocio.

Además debe contar con habilidades como saber elaborar diagramas UML y Mapa de Procesos con herramientas CASE. Debe saber además describir brevemente los procesos del negocio y elaborar los diagramas de dichos procesos empleando los diagramas de actividad de UML, así como describir las actividades implícitas en estos procesos y las entidades conceptuales. También debe expresarse con claridad y empleando correctamente el lenguaje de ideas oralmente y redactar ideas claras cuidando de las reglas ortográficas. Otras habilidades un poco más avanzadas que podría mejorar la calidad de su trabajo serían, diseñar y mejorar los procesos de negocio y elaborar diagramas de procesos de negocio empleando BPMN y patrones de workflow.

3.4.2 Arquitecto de Integración

El arquitecto de Integración trabaja conjuntamente con el de sistema puesto que sus actividades están estrechamente relacionadas.

Responsabilidades

Dentro de las responsabilidades del Arquitecto de Integración se encuentran las de: establecer y normalizar la comunicación, estableciendo cómo se va a llevar a cabo la interacción entre componentes. Debe analizar y proponer soluciones de integración con aplicaciones externas.

El arquitecto de integración establece, norma y justifica cada uno de los elementos de integración. Se encarga de documentar, planificar, formalizar y evaluar la integración a todos los niveles y dimensiones.

Estas son las actividades generales que realiza. Profundizando un poco más, este arquitecto tiene tareas como el análisis de los componentes horizontales que se establezcan, identifica los flujos existentes en la arquitectura de negocio, cuya trazabilidad a la arquitectura del sistema deberá ser gestionada y controlada por el equipo de integración. Identifica los puntos de acceso a recursos concurrentes, las reglas pre y pos condicionales, así como la notificación de eventos y trazas de los mismos asociadas a las acciones. Identifica formatos y estándares que se aplicarán. Además

formaliza los recursos de integración, los nodos de integración, los flujos de trabajo estáticos, formaliza un grupo de acciones dinámicas controladas, y también planifica la integración. Todas estas actividades son explicadas y argumentadas en la Vista de Integración de la Arquitectura.

Otros elementos a tener en cuenta que condicionan las responsabilidades del arquitecto de integración son:

- Colaborar en la comprensión de requerimientos funcionales y de calidad para las aplicaciones integradas.
- Crear el proyecto de arquitectura inicial para las aplicaciones integradas.
- Seleccionar tecnologías de integración adecuadas que cumplan con los requerimientos de la aplicación.
- Confirmar que la combinación de la arquitectura y la tecnología de integración que se utilizan para construir la aplicación para toda la empresa pueden ser exitosas antes de realizar una gran inversión para su implementación.

Competencias del Arquitecto de Integración.

Para poder desempeñarse en tantas actividades el arquitecto de integración, además de tener todas las competencias de todo Arquitecto de Software, debe tener amplios conocimientos en patrones de diseño arquitectónicos y aplicar patrones de integración. Tiene que poder identificar niveles de integración e integrarlos a las diferentes tecnologías implicadas. Le compete conocer y definir los protocolos de comunicación entre soluciones y debe saber formalizar los elementos asociados. Ordena y guía el desarrollo de las soluciones de nodos de integración y motiva al equipo a llevar a cabo los objetivos propuestos, consolida su enfoque en implementar la especificación definida. Este arquitecto debe estar en constante interacción con las demás disciplinas porque cualquier cambio en una de ellas puede afectar la integración definida.

3.4.3 Arquitecto de Datos

El arquitecto de datos es la persona encargada de manejar y controlar todo lo referente a la Base de Datos, ya sea su persistencia, integralidad y funcionalidad. Son los encargados de la estructura, complejidad y flexibilidad de la base de datos.

Responsabilidades

Los objetivos de este rol se centran en definir una política de integración y objetivos de trabajo en función del Centro de Datos, definir una metodología de implementación de réplica, documentar la solución, definir método de actualización y control de versiones de la base de datos, definir la metodología de trabajo de la arquitectura de datos y la estrategia de soporte, crear el Expediente de la vista de datos y un Plan de formación, así como definir métricas e indicadores.

El arquitecto de datos es imprescindible en el desarrollo de software, evidenciando esto se presentan las principales responsabilidades que tiene:

- Diseña las soluciones de datos así como las de integración.
- Construye la vista de datos, también define y dirige la arquitectura de datos y las soluciones, metodológica y técnicamente.
- Conforma, construye y mantiene el script de instalación de la BD, gestiona los servicios de la BD y coordina el trabajo con ella.

Los arquitectos de datos ayudan a los arquitectos de software a diseñar la estructura, complejidad y flexibilidad de la base de datos usando los principios WYSIWYG[16] en la construcción de los diagramas entidad-relación. Los arquitectos de datos pueden comprobar el modelo de la BD para asegurarse que no contiene errores antes de generar el script de la BD. Cuando se comprueba el modelo, el programa verifica que la base de datos no contiene datos duplicados. Además genera las declaraciones del SQL que creará su base de datos en el servidor de la base de datos y lo salvarán en un archivo del disco o lo ejecutarán contra una ODBC* (Open Data *Base* Connectivity) fuente, mientras creando tablas, índices y referencias en la base de datos. Éste guarda las especificaciones

[16] **WYSIWYG** es el acrónimo de ***W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et* (en inglés, "lo que ves es lo que obtienes"). Se aplica a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso.

de formato de base de datos (el tipo de datos, la creación del tablas/referencias/índices, comandos de creación, etc.) en los archivos del texto externos que le permiten no sólo editar/actualizar las especificaciones de formato, sino incluso agregar las nuevas descripciones de la base de datos al programa. Al crear una base de datos del modelo, los usos del programa se unificaron a drivers de ODBC para conectar al servidor de la base de datos.

Para poder llevar a cabo su trabajo depende del analista, ya que necesita de una captura de requisitos, análisis y diseño.

RUP propone la aparición del arquitecto de datos en la etapa de Análisis y Diseño donde comienza con el modelo de datos.

Los autores de este trabajo coinciden con esta propuesta, apareciendo este arquitecto en la etapa de diseño, después que están hechos los prototipos de interfaz de usuario, los requerimientos y las descripciones de los casos de uso al modelar la solución. Si lo incluimos antes puede haber cambios en el análisis o diseño que conlleven a grandes cambios y pérdida de tiempo en lo referente a la persistencia y diseño del Modelo de Datos.

Competencias del Arquitecto de Datos.

La persona que desempeñe el Rol de arquitecto de Datos debe poseer amplios conocimientos en la teoría relacional de Base Datos (BD) (cálculo, álgebra y normalización.) Debe saber de las diferentes herramientas de modelado para BD, tanto las de creación de una BD como Postgre, SQL, MSQL, Oracle, como las de modelado ER/Studio, Visual Paradigm*, y aunque no domine el trabajo al cien por ciento de todas ellas si debe estar al día con las existentes y tener una idea de cómo se utilicen. Es lógico pues que deba saber sobre diseñar base de datos, MER, normalización y desnormalización. Debe saber sobre la arquitectura del gestor de base de datos que está utilizando. También tener conocimientos sobre optimización, conocimientos de SQL estándar, etc. Dirigir y liderar es otra de las características principales de este arquitecto, mas cuando se hable de un proyecto de envergadura donde exista un equipo completo de datos. Debe ser una persona responsable con sus tareas, estudioso y que se mantenga actualizado siempre en el ámbito de las base de datos.

3.4.4 Arquitecto Temático

Responsabilidades

El Arquitecto Temático como bien se describía antes, tiene las mismas responsabilidades que el Arquitecto de Sistema, lo que enfocado a una única línea temática. Este arquitecto identifica los componentes, establecer conectores, interfaces claras, identificar soluciones del sistema, nodos críticos de integración, definir variables globales, la línea Base, los estándares y normas y prioriza componentes, todo igual que su homólogo, pero todo esto lo hace solamente para el subsistema en el que trabaje.

Competencias del Arquitecto Temático

Como puntos imprescindibles debe contar con conocimientos sobre estilos arquitectónicos, patrones de diseño, metodologías de desarrollo. Y todos los demás conocimientos de liderazgo, políticas organizacionales, etc., que tiene el Arquitecto de Sistema.

Conclusiones del capítulo

Con este capítulo se ha cumplido el objetivo propuesto de dar a conocer los diferentes roles que los autores de este trabajo estiman imprescindibles a la hora de desarrollar una Arquitectura de Software, viendo su importancia dentro de esta disciplina, las diversas responsabilidades que poseen así como las competencias o habilidades necesarias para cumplir con estos roles.

Capítulo 4. Expediente de la Arquitectura

Introducción del Capítulo

Todo flujo de trabajo que se desarrolla durante la construcción de la arquitectura, debe estar organizado y/o estructurado con el objetivo de crear un lugar en el que depositar esta información organizada. Un lugar que constituya un espacio para documentación, que evite sobrescribir documentos, código etc., que lleve un control de versiones y que guíe el proceso de desarrollo del software en general. El expediente tecnológico es una carpeta destinada a este fin.

Expediente de la Arquitectura de Software

Este expediente consta de una carpeta para cada una de las vistas de la arquitectura propuestas en el Capítulo 2 y una carpeta para recoger todos los aspectos relacionados con la gestión de la calidad debido a que en cada una de las áreas se deben definir estándares y listas de chequeo a utilizar por las demás líneas de desarrollo.

Ilustración 31. Expediente de la Arquitectura de Software

Dentro de cada una de estas carpetas se encuentra la documentación asociada a cada disciplina, organizada en una serie de subcarpetas que tienden a ser comunes para cada arquitectura. A continuación se irá mostrando por cada una de ellas los artefactos que contienen y la descripción de cada uno. Se aclara que los nombres de las carpetas y artefactos que contiene el expediente no tienen tilde debido a las distintas codificaciones que tienen los sistemas operativos, y de esta forma evitar posibles problemas de este tipo.

Expediente Arquitectura de Datos

Ilustración 32. Expediente de la Arquitectura Datos

La carpeta **Documentos de soporte** contiene los documentos creados por el equipo de datos para apoyar sus soluciones. Esta puede contener la matriz de integración que se describe en la vista de datos o cualquier otro tipo de artefacto generado en este sentido.

La carpeta llamada **Integración de Datos** contiene el documento de integración de datos el cual define cómo se realiza la integración de los diferentes subsistemas y módulos a nivel de datos. Este documento rige el proceso de integración indicando qué componentes entran y salen de cada subsistema, especificando cómo queda modelada esta integración en la base de datos y modelando por cada subsistema cómo se integra a los demás a ese nivel. Además en este documento se realiza una matriz de traza de datos por cada subsistema que resulta de gran ayuda para la integración ya que especifica detalladamente los datos de entrada y de salida de los subsistemas, por cada subsistema se declara con cual otro se integra especificando la tabla origen, la tabla destino, la relación entre las tablas y la variable que será objeto de esta integración. A este documento también se le pueden agregar otros aspectos que se consideren importantes para la integración por parte del equipo de arquitectura de datos, además de los mencionados.

Luego, la carpeta **Modelo de Datos** contiene los modelos de datos de cada uno de los subsistemas y módulos del sistema en general. Por cada módulo se tienen 3 subcarpetas, como se muestra en la Figura 31, una que contiene un diccionario de datos, que es la descripción de todos los objetos de la base de datos; otra para el modelo de la base de datos que recoge el archivo con la base de datos modelada y por último una subcarpeta en la que se guarda una imagen de este mismo modelo para facilitar su visualización sin tener que ejecutar la herramienta con que se modeló.

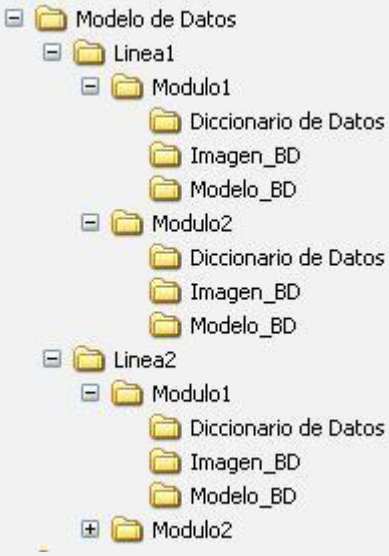


Ilustración 33. Modelo Datos

En la carpeta **Pruebas de conceptos** están contenidas las pruebas realizadas directamente a la base de datos como son la prueba de rendimiento, que implica un trabajo alterno con la prueba de tamaño de la base de datos. La prueba de concurrencia a la base de datos es el acceso a ella de varios usuarios al mismo tiempo sin tener fallas en las peticiones, además otro aspecto a tener en cuenta es el del respaldo o seguridad de la base de datos, entre otros que deben ser documentados como constancia de los chequeos al sistema.

El documento de **Soluciones de la Arquitectura de Datos** se encuentra en la carpeta **Soluciones de la Arquitectura de Datos**, en este se reflejan todas las soluciones implementadas por el equipo de arquitectura de datos, dando primeramente una panorámica de la situación que originó la implementación de la misma, luego detallando la solución implementada y finalmente se refleja la prueba de concepto realizada a esta solución y su resultado.

La **Vista de datos** contiene en su interior el documento de Vista de Arquitectura de Datos, este documento constituye la guía para el desarrollo de la base de datos desde el punto de vista arquitectónico, con el propósito de estandarizar el desarrollo de todas las líneas, generalizando patrones, herramientas, nomenclaturas y otros aspectos que mejoran el rendimiento de la base de datos. Aquí se definen las políticas de trabajo en cuanto al desarrollo de la BD, es decir, políticas para la integración entre líneas, para la creación de índices, el control de cambios, para el uso de UPDATE y DELETE en CASCADA, para el uso de los tipos de datos teniendo en cuenta el uso de réplicas entre bases de datos y además para el uso de reglas de nomenclatura y estándares definidos para los objetos de la base de datos. De igual forma se indica cómo se garantiza la seguridad en la base de datos, especificando qué gestor y qué servidores de base de datos utilizar,

cómo realizar el respaldo de la misma, qué usuarios tiene la base de datos y qué permisos tiene cada uno. Para asegurar un código más legible y reutilizable, de manera que se pueda aumentar su mantenibilidad a lo largo del tiempo, en este documento también se establecen las pautas para normar el comentariado en la BD. Igualmente se estandariza la nomenclatura que se utiliza en la creación de tablas, esquemas, base de datos, campos, llaves primarias, llaves foráneas, secuencias, funciones, triggers, tipos de datos, vistas, dominios, etc.

La vista de datos declara además qué herramientas se usan en el diseño y la implementación de la BD. Describe cómo son representados los diferentes tipos de datos (Numeric, Date, Varchar, etc.), cómo se realizará la generación de llaves primarias, la concurrencia y la normalización de árboles. En este orden también se define la política de indexado por la que se regirá la base de datos y se regula como se asegura el rendimiento de la misma indicando aspectos que pueden mejorar dicho rendimiento, como son cambios en el diseño de las tablas o de las consultas y la reducción del tamaño. La normalización y la desnormalización de la base de datos también se definen aquí a fin de guiar a los implementadores de la BD en este sentido. Otro aspecto que debe quedar plasmado en este documento es cómo se realiza la distribución a nivel de datos que resulta de gran ayuda al equipo de implantación para realizar el despliegue del sistema.

Expediente Arquitectura de Integración

El expediente de Arquitectura de Integración recoge todos los artefactos generados en esta área relacionada con la planificación, formalización y especificación de los elementos de integración a todos los niveles y dimensiones. Se estructura como se muestra en la figura 32.

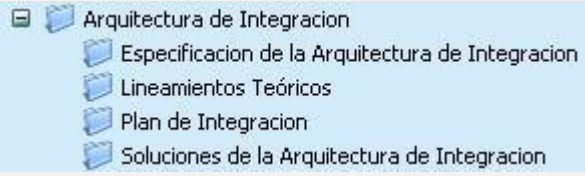


Ilustración 34. Expediente de la Arquitectura Integración

En la carpeta **Especificación de la Arquitectura de Integración** se realiza la formalización de la integración de los subsistemas mediante la representación de las vistas de integración, la interoperabilidad entre subsistemas y componentes. Además se especifican las formas, métodos y formatos de la integración, es decir, cómo se integran.

La carpeta **Lineamientos Teóricos** de la arquitectura de integración contiene el documento con el mismo nombre que recoge los aspectos teóricos que fueron utilizados en esta área, se especifican los niveles y tecnologías, y además las formas y mecanismos de integración que se le aplican a cada subsistema. También se realiza una justificación de los patrones de integración y estilos arquitectónicos utilizados en la integración de los subsistemas.

Durante la planeación de la integración se realiza una especificación condicionada por los niveles. Una vez identificado el nivel o niveles de integración que se van a aplicar a una aplicación o subsistema determinado, se planifica cómo integrar los elementos (aplicaciones, subsistemas, módulos y componentes) implicados, para cada uno se especifica roles, artefactos de entrada y artefactos de salida. Esta planificación se recoge en el **Plan de Integración** y se define uno para cada subsistema asociado a cada iteración que se especifica en el Plan de Iteración de la vista de sistema. A partir de la matriz de integración y el modelo de componentes se realiza según las dependencias y la priorización de los componentes un cronograma de cumplimiento de las integraciones.

Las **Soluciones de la Arquitectura de Integración** se describen en el documento de Soluciones detallando todos los componentes implementados. De cada uno de estos componentes se especifica la situación que da origen a la implementación del mismo, además se describe el modelo conceptual y la solución técnica asociada a los patrones aplicados. Esta carpeta contiene además la **Plantilla de especificación de interfaces** que muestra como se deben implementar las interfaces.

Expediente Arquitectura de Presentación

El expediente tecnológico de la Arquitectura de Presentación está estructurado como se representa en la figura 33.

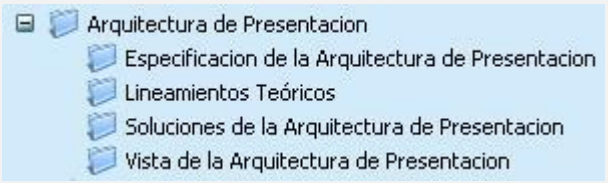


Ilustración 35. Expediente de la Arquitectura de Presentación

En las **Especificaciones de la Arquitectura de Presentación** se define la nomenclatura de los componentes con sus datos descriptivos así como los datos asociados a las etiquetas. Según el framework utilizado se definen de los componentes las propiedades mínimas e indispensables que

se usan en el desarrollo de la arquitectura de presentación. Se validan los componentes y si existe alguna validación entonces se enumeran las expresiones regulares disponibles. Finalmente se describen las funciones.

En el documento **Lineamientos teóricos** se recogen los principios arquitectónicos que se definen teóricamente y que constituyen la línea base para definir el diseño de la arquitectura de presentación.

Otro de los documentos en esta carpeta general es el de **Soluciones de la Arquitectura de Presentación**. Este documento refleja la situación planteada, a partir de la cual se desarrollan conceptualmente algunas soluciones y por último se enumeran las soluciones definitivas implementadas.

La **vista de presentación** es el documento en el que se argumentan los aspectos bases que guían la arquitectura de presentación como son la forma en que se desarrollarán las interfaces, las validaciones, la utilización de los CSS y otros aspectos que considere el equipo de arquitectura de presentación.

Expediente Arquitectura de Seguridad

El expediente tecnológico de Arquitectura de Seguridad se realiza con el objetivo de agrupar los artefactos generados en esta área.

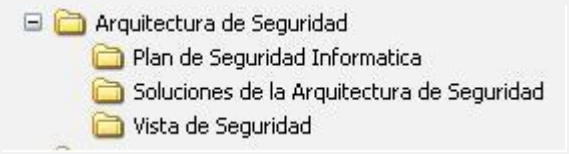


Ilustración 36. Expediente de la Arquitectura de Seguridad

El **Plan de Seguridad Informática** constituye el instrumento básico para lograr la confidencialidad, integridad y disponibilidad de la información y la protección de los medios y locales donde se utilice la técnica de computación. En este documento se recoge la caracterización del sistema informático, se describen las amenazas y riesgos detectados en el ambiente de desarrollo y despliegue del software, se formulan las políticas y medidas de seguridad informática a cumplir en el centro. Se describe cómo se implementan, en las áreas a proteger, las políticas generales que han sido definidas para toda la entidad, en correspondencia con las necesidades de protección en cada una de ellas, atendiendo a sus formas de ejecución, periodicidad, personal participante y medios.

En el documento **Soluciones de la Arquitectura de Seguridad** se recogen las soluciones que define el equipo de Arquitectura de Seguridad para proteger la información, como puede ser la implementación de sistemas y componentes.

El documento de **Vista de Seguridad** sirve de guía para el desarrollo, y norma todos los aspectos a tener en cuenta en cada fase del desarrollo del software para garantizar la seguridad del mismo. Se establecen las pautas a seguir para garantizar la seguridad en el entorno de desarrollo del software, la seguridad durante el desarrollo de las aplicaciones, la seguridad en el entorno de despliegue y la seguridad de las aplicaciones propiamente.
Este documento también debe especificar los protocolos de comunicación segura, mecanismos de encriptación de datos, mecanismo de llave pública y llave privada (PKI), el grado de fortaleza de las claves de los usuarios del sistema y de la base de datos que se utilizará. Así como se describirá el sistema o componente implementado en pos de garantizar la seguridad de la aplicación en sí, es decir que garantice la autenticación y autorización de la aplicación.

Expediente Arquitectura de Sistema

El Expediente de la Arquitectura de Sistema recoge la documentación formal e imprescindible con la que se debe contar para la creación de la Arquitectura de Sistema.

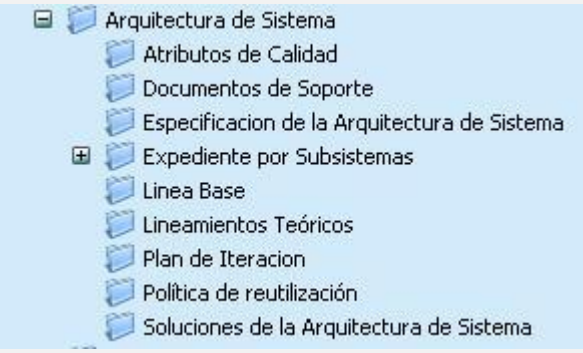


Ilustración 37. Expediente de la Arquitectura de Sistema

Dentro de estos documentos están:

Atributos de calidad, cuando se refiere a atributos de calidad de la Arquitectura de Sistema (Arquitectura de Software) se refiere a los requisitos no funcionales a los que esta da cumplimento. En este sentido la Arquitectura de Software debe cumplir con los atributos referentes a mantenimiento, rendimiento y reutilización del software. Este documento describe como la Arquitectura de Software cumple con estos atributos de calidad.

En este expediente se crea la carpeta de **Documentos de soporte** en la que se recogerán los artefactos complementarios que se generen en esta área como puede ser el documento de la Priorización de Requisitos, la Matriz de Integración y el documento Variables Globales. En el documento de Priorización de Requisitos se hace una estimación de cada requisito en cuanto a tiempo y complejidad brindando una idea de cuáles deberían comenzarse a implementar primero y cuáles de último. La Matriz de Integración se realiza mediante un Excel con la relación de lo que necesita cada componente y de quién lo necesita por cada uno de los subsistemas o módulos del sistema en general. El documento Variables Globales especifica las variables o conceptos globales que utilizan los subsistemas.

Uno de los artefactos más importantes que se generan en esta área se recoge en la carpeta **Especificación de la Arquitectura de Software**, esta contiene el documento de Especificación General de la Arquitectura del Sistema. Este documento tiene como objetivo describir general y detalladamente todos los aspectos de la arquitectura de sistema, como la representación arquitectónica, las dimensiones de la integración, la estructura de los componentes, todos los subsistemas en los que estará dividido, entre otros temas.

El **Expediente de los Subsistemas** recoge los artefactos que se deben generar en cada subsistema, estos son documento de identificación y descripción de los componentes, diagrama de componentes documentado, diagrama de clases de diseño de cada componente, documento de Integración de sistema y datos y el documento de especificación de la integración del subsistema.

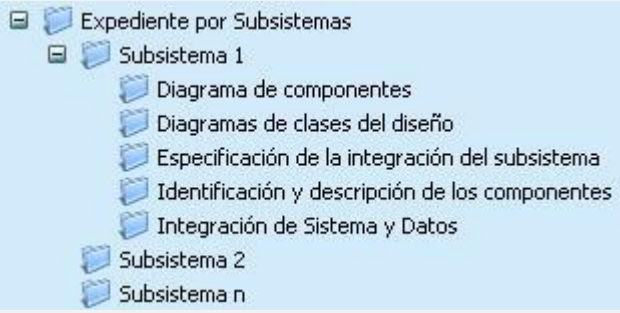


Ilustración 38. Expediente por subsistema

Diagrama de clases de diseño de cada componente, en este documento se debe realizar por cada componente el diseño de clases correspondientes tomando como base la del marco de trabajo propuesto así como la arquitectura vertical descritos en otros documentos, en el mismo deberán quedar plasmadas las clases presentes en cada una de las partes del Modelo Vista Controlador (MVC)* propuesto, con las principales funciones y la relación y colaboración entre ellas.

Diagrama de componentes documentado, en este documento se debe establecer el diagrama de todos los componentes identificados, documentando en qué consisten y todas las responsabilidades de cada componente, así como las correspondientes interfaces de comunicación que tengan. En cada interfaz deben quedar especificados todos los contratos que deben brindar esos componentes y quiénes los van a consumir.

El documento de especificación de la integración del subsistema especificará en el formato y bajo la plantilla seleccionada y explicada en otro documento los servicios que el subsistema necesita consumir especificando una serie de atributos y características de cada uno de estos contratos, así como también los servicios que debe proveer el sistema.

En el **documento de identificación y descripción de los componentes** es donde deben quedar registrados todos los componentes identificados, y por cada uno de estos componentes identificar y plasmar los requisitos con los que se corresponden y la complejidad de estos. También debe tener la cantidad de tablas en la base de datos con las que interactuará cada componente, así como la cantidad de entidades de negocio y clases gestoras presentes en el componente, obteniendo como resultado final, la complejidad del componente.

En el **documento Integración de sistema y datos** debe quedar relacionado por cada componente las tablas del modelo de datos con las cual este va a interactuar, describiendo las principales transacciones sobre estas tablas que realizará el componente.

Otro artefacto dentro del expediente de la Arquitectura de Sistema que resulta de gran importancia es la **Línea Base**, este documento permitirá guiar y evaluar el desarrollo del proyecto, según las normas definidas por la Arquitectura, y además permitirá puntualizar los elementos de configuración del software.

Lineamientos teóricos: Este es el documento de las decisiones teóricas de la arquitectura. Debe definir estilos arquitectónicos a utilizar, principios de arquitectura que se aplican, tendencias, corrientes de desarrollo. Además podría tener el alcance de este diseño arquitectónico y posibilidades de crecimiento.

También es en esta área donde se realiza el **Plan de Iteración** el cual tiene como objetivo definir detalladamente para cada una de las iteraciones a realizarse el orden en el que los componentes serán integrados.

Otra de las carpetas **Política de Reutilización** contiene las políticas de reutilización de los componentes indicando cómo se realizarán las mismas en los diferentes subsistemas.

Soluciones horizontales: es el documento que recoge las soluciones que se definieron y que son necesarias para crear un producto configurable y que pueda ser aplicado a cualquier sistema contable. Es el documento que en buena lid justifica todas las decisiones que se toman con respecto a los componentes que darán solución a problemas generales del sistema.

Expediente Arquitectura de Tecnología

La carpeta Arquitectura Tecnológica contiene los artefactos que genera esta disciplina, estructurada como se muestra en la figura 37.

Ilustración 39. Expediente de la Arquitectura de Tecnología

Los aspectos legales de las tecnologías utilizadas y creadas durante el desarrollo del software se agrupan en la carpeta **Especificación Legal** que dentro de ella contiene las licencias de las tecnologías utilizadas, la licencia del marco de trabajo desarrollado y la vista legal de la arquitectura.

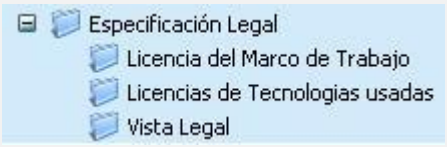


Ilustración 40. Especificación Legal

El documento **Vista Legal** de la arquitectura refleja todas las tecnologías y software utilizados en el desarrollo y despliegue del producto. De cada uno de ellos se especifica nombre, versión, fabricante, licencia, plataforma, categoría y costo. Además de esto el documento debe contener el dictamen legal emitido por el abogado de la empresa o institución en la que se desarrolle el software.

Toda la **especificación técnica** se recoge en la carpeta con este mismo nombre y ésta a su vez se divide en 2 subcarpetas, una para la documentación técnica de los componentes y otra para la especificación de la vista tecnológica.

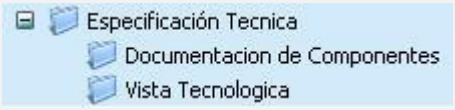


Ilustración 41. Especificación Técnica

La **documentación técnica de los componentes** se realiza mediante la descripción de los requisitos funcionales y no funcionales del componente, la descripción del diseño del mismo, es decir teniendo en cuenta el diagrama de clases del diseño, los diagramas de secuencia, modelo de datos y diagrama de componentes. También se realiza la especificación y descripción de los elementos de configuración del componente, ficheros de configuración .XML, YML, .php, .conf, .ini y además una descripción y especificación de los casos de prueba, casos de estudio y patrones arquitectónicos utilizados en la implementación del componente.

El documento **vista tecnológica** de forma general muestra todo el proceso que se lleva a cabo para la confección del marco de trabajo mediante la selección de los escenarios, plataformas, frameworks y librerías que conformarán el mismo. Esta selección se realiza mediante la evaluación de las características que se desean obtener. Después de haber realizado la selección se describe la arquitectura del framework confeccionado y se realiza la representación gráfica del mismo. Por último este documento debe contener las pruebas de concepto realizadas al framework.

La carpeta **Manuales de MT (Marco de Trabajo)** recoge los artefactos generados en ara de facilitar el trabajo de los usuarios con el marco de trabajo, estos artefactos pueden ser el manual de usuario, el manual de instalación y casos de estudio.

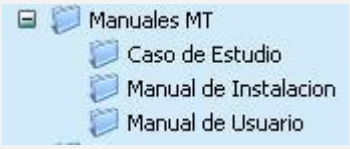


Ilustración 42. Manuales MT

El manual de usuario del MT es una guía de todos los pasos y procedimientos que se deben seguir para alcanzar resultados óptimos en cuanto al desarrollo de aplicaciones con el marco de trabajo. Debe explicar cómo crear los proyectos, los módulos y submódulos, debe contener además una explicación de los temas de configuración con ejemplos donde se expliquen. Además de esto debe figurar también en él cómo instalar y configurar cada una de las herramientas requeridas para el trabajo con la arquitectura.

En la carpeta **Caso de Estudio** se deben describir algunos casos de estudio para que los usuarios puedan comprender la utilización del marco de trabajo mediante ellos. Cada caso de estudio debe mostrar de forma detallada al lector de explicaciones del trabajo con el caso de estudio, se debe aclarar todo el flujo de información del componente. Con esto se persigue el objetivo de lograr en los programadores del proyecto un mejor conocimiento sobre los componentes del framework y su integración a los subsistemas del proyecto, además se persigue la obtención de un aprendizaje superior sobre el marco de trabajo, su configuración y el estándar que se persigue por parte de la arquitectura y por supuesto un mayor alcance sobre el lenguaje de programación que se define.

El **manual de instalación** describe como llevar a cabo la instalación del marco de trabajo en una PC local para luego poder utilizarlo.

La carpeta **Paquetes estables del MT** contiene el código fuente del marco de trabajo separado por las versiones desarrolladas y explica cómo se lleva a cabo este control de versiones en el **Documento de control de versiones**. La carpeta **casos de estudio** contiene las fuentes de los casos de estudios explicados en los manuales del marco de trabajo.

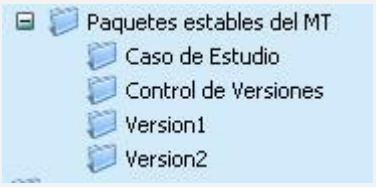


Ilustración 43. Paquetes estables del MT

Expediente para la Gestión de la Calidad

Esta carpeta se crea con el objetivo de recoger los aspectos relacionados con la gestión de la calidad durante la construcción y mantenimiento de la arquitectura del software. La misma está compuesta por 2 subcarpetas, la primera se destina a agrupar todas las listas de chequeo que se conciben en las áreas para garantizar que se cumplan con los aspectos normados en cada una de ellas. Estas no son más que un listado de preguntas, en forma de cuestionario que sirve para verificar el grado de cumplimiento de determinas reglas establecidas a priori con un fin determinado. (Bichachi, 2009)

La segunda carpeta se destina a agrupar todos los estándares establecidos ya sea estándares de código, de documentación, de interfaces, de nomenclatura, etc. que son muy necesarios implementarlos durante la construcción de una Arquitectura de Software.

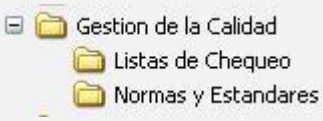


Ilustración 44. Gestión de la Calidad

Conclusiones del Capítulo

Se ha abordado en este capítulo una de las maneras en que se puede organizar la documentación de cualquier proyecto de Arquitectura de Software en una estructura de carpetas. Siendo necesaria e importante la organización y estructuración de los artefactos obtenidos en el proceso de desarrollo.

Capítulo 5. Pruebas de Concepto de la Arquitectura

Introducción del Capítulo

No se puede hablar de la Arquitectura de Software sin tener presente la calidad de la misma. Obtener una arquitectura robusta, depende entre otras cosas, de las pruebas de concepto que se le apliquen. En este capítulo se definen las pruebas de concepto, algunas técnicas y métodos de evaluación, quiénes los propusieron, así como el estado de estas pruebas y las principales herramientas existentes hoy día en el mundo para aplicarlas.

¿Por qué evaluar una arquitectura?

Todos los diseños arquitectónicos implican desventajas en las cualidades del sistema, ya que estas dependen en gran medida de las decisiones arquitectónicas; por lo que garantizar la calidad del sistema es a menudo a expensas de garantizar calidad en la arquitectura.

La arquitectura es el primer artefacto del ciclo de vida del desarrollo de un software que incorpora importantes decisiones de diseño; las decisiones son fáciles de tomar, pero difíciles de cambiar una vez que el sistema se aplica.

5.1 Pruebas de concepto.

Las pruebas de concepto, o evaluación de la arquitectura de un sistema de software, consisten en el análisis de la estructura para identificar los riesgos potenciales y verificar que los requisitos de calidad se han tenido en cuenta en el diseño. En otras palabras es la evaluación de las decisiones arquitectónicas o evaluación de la funcionalidad de la utilización de un conjunto de tecnologías para resolver un problema determinado.

El propósito de realizar estas pruebas de concepto o validación es demostrar que el producto y sus componentes cumplen con los requerimientos especificados y que el funcionamiento sea correcto en el ambiente propuesto.

Existen varios tipos de Pruebas de Concepto (PoC) según lo que se quiera probar. Por ejemplo, si lo que se pretende es testear de antemano el rendimiento de la aplicación, o su escalabilidad, la prueba se limita a estresar un entorno de servidores similar al que se planee para la Producción, a efectos de conocer los límites o puntos de quiebre de la aplicación (cuántos usuarios concurrentes, cuántos requerimientos simultáneos será capaz de atender con un tiempo de respuesta no mayor a los 200

ms, etc.) Otros tipos de PoC pueden testear la seguridad de la aplicación simulando ataques y observando cómo reacciona el sistema, y así sucesivamente. La Prueba de Concepto es una etapa necesaria para validar la arquitectura, para decidir si seguir adelante o someterla a ajustes o revisiones posteriores. Pero fundamentalmente, es un elemento que al arquitecto le debe proveer seguridad en sus decisiones, y a la vez inspirar confianza en el resto de los interesados en la aplicación.

Por tanto las finalidades de las pruebas de concepto son:

- Decidir si el proyecto se lleva a término.
- Confirmar la elección del concepto en la solución propuesta.
- Abordar ideas de mejora.
- Verificar si ya se encuentra en una etapa para el Despliegue.

Y como resultados se pueden obtener:

- Viabilidad del despliegue de la solución en un ambiente similar al requerido
- Validación de las funcionalidades de la solución
- Adecuar y afinar los elementos de software y hardware requeridos
- Validación de las herramientas de almacenamiento
- Determinar las posibles vulnerabilidades que puedan existir y definir la vía de mitigación de dichas vulnerabilidades y riesgos

La evaluación de la Arquitectura de Software puede ser realizada mediante el uso de diversas técnicas y métodos. En este sentido, resulta interesante estudiar las distintas opciones que existen en la actualidad para llevar a cabo esta tarea.

5.2 Técnicas de evaluación.

El interés se centra en determinar el momento propicio para efectuar la evaluación de una arquitectura. En este sentido, Kazman amplía el panorama clásico, indicando las ocasiones en que se hace posible hacer la evaluación de una arquitectura. Su planteamiento establece que es posible realizarla en cualquier momento, pero propone dos variantes que agrupan dos etapas distintas: temprano y tarde.

Para la primera variante, Kazman establece que no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

La segunda variante propuesta por Kazman consiste en realizar la evaluación de la arquitectura cuando ésta se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado. Los autores consideran muy útil la evaluación del sistema en este punto, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema, y cómo será su comportamiento general.

Por su parte, Bosch afirma que la evaluación de una Arquitectura de Software es una tarea no trivial, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. Las mediciones que se realizan sobre una Arquitectura de Software pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Los objetivos que menciona Bosch son tres: *cualitativos*, *cuantitativos* y *máximos y mínimos teóricos*.

La medición *cualitativa* se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle.

La medición *cuantitativa* busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una Arquitectura de Software. El esquema general es la comparación con márgenes establecidos, como lo es el caso de los requerimientos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella. Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación.

Por último, la *medición de máximo y mínimo teórico* contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

En este sentido, Kazman afirma que de la evaluación de una arquitectura no se obtienen respuestas del tipo “si - no”, “bueno – malo” o “6.75 de 10”, sino que explica **cuáles son los puntos de riesgo del diseño evaluado.**

Una de las diferencias principales entre los planteamientos de Bosch y Kazman es el enfoque que utilizan para efectos de la evaluación. El método de diseño de arquitecturas planteado por Bosch tiene como principal característica la evaluación explícita de los atributos de calidad de la arquitectura durante el proceso de diseño de la misma. Afirma que el enfoque tradicional en la industria de software es el de implementar el sistema y luego establecer valores para los atributos de calidad del mismo. Este enfoque, según su experiencia, tiene la desventaja de que se destina gran cantidad de

recursos y esfuerzo en el desarrollo de un sistema que no satisface los requerimientos de calidad. En este sentido, Bosch plantea las técnicas de evaluación: basada en escenarios, en simulación, en modelos matemáticos y en experiencia. Por su parte, Kazman propone que resulta de poco interés la caracterización o medición de atributos de calidad en las fases tempranas del proceso de diseño. Su enfoque se orienta hacia la mitigación de riesgos, ubicando dónde un atributo de calidad de interés se ve afectado por decisiones arquitectónicas.

Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Por ejemplo, un usuario hará la descripción en términos de la ejecución de una tarea; un encargado de mantenimiento hará referencia a cambios que deban realizarse sobre el sistema; un desarrollador se enfocará en el uso de la arquitectura para efectos de su construcción o predicción de su desempeño. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta.

El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc.

El contexto describe qué sucede en el sistema al momento del estímulo.

La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado.

Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. Kazman y Carriere (19) coinciden en la importancia del uso de los escenarios, no sólo para efectos de la evaluación de arquitecturas de software.

Entre las ventajas de su uso están:

- Son simples de crear y entender
- Son poco costosos y no requieren mucho entrenamiento
- Son efectivos

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes, a saber: el Árbol de utilidades propuesto por Kazman y los perfiles, propuestos por Bosch.

Árbol de Utilidades

Según Kazman, un *Árbol de Utilidades* es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. La intención del uso del Árbol de Utilidades es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol.

El Árbol de Utilidades contiene como nodo raíz la *utilidad general* del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol, los cuáles se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

Perfiles.

Un perfil es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados. Los *perfiles completos* definen todos los escenarios relevantes como parte del perfil. Esto permite al Ingeniero de Software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, puesto que incluye todos los posibles casos. Su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad.

Los *perfiles seleccionados* se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo a algunos requerimientos. La aleatoriedad no es totalmente cierta por limitaciones prácticas, por lo que se fuerza la realización de una selección estructurada de elementos para el conjunto de muestra. Si bien es informal, permite hacer proposiciones científicamente válidas.

Evaluación basada en simulación

Bosch establece que la evaluación basada en simulación utiliza una implementación de alto nivel de la Arquitectura de Software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación –a cierto nivel de abstracción- del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad.

Evaluación basada en modelos matemáticos

Bosch establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro.

Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes, y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales. Entre los instrumentos que se cuentan para las técnicas de evaluación de arquitecturas de software basada en modelos matemáticos, se encuentran las Cadenas de Markov y los Diagramas de bloque de fiabilidad.

Evaluación basada en experiencia

Bosch establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño, aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación.

Existen dos tipos de evaluación basada en experiencia: la *evaluación informal*, que es realizada por los Arquitectos de Software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

5.3 Métodos de evaluación

Kazman propone que la existencia de un método de análisis de arquitecturas de software hace que el proceso sea repetible, y ayuda a garantizar que las respuestas correctas con relación a la arquitectura pueden hacerse temprano, durante las fases tempranas de diseño. Es en este punto donde los problemas encontrados pueden ser solucionados de una forma relativamente poco costosa. De manera similar, un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. Por esta razón, resulta conveniente estudiar los métodos de evaluación de arquitecturas de software propuestos hasta el momento. Su enfoque se orienta hacia la mitigación de riesgos, ubicando dónde un atributo de calidad de interés se ve afectado por decisiones arquitectónicas. En su estudio, Kazman presenta tres métodos de evaluación de arquitecturas de software, que son Método de Análisis de Desventajas de la Arquitectura (Architecture Trade-off Analysis Method, ATAM), Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) y Revisión de Diseño Activo Intermedio (Active Intermediate Designs Review, ARID).

Método de Análisis de Arquitecturas de Software (SAAM)

Según Kazman el Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. De acuerdo con Kazman las salidas de la evaluación del método SAAM son las siguientes:

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.
- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

Método de Análisis de Desventajas de Arquitectura (ATAM)

Según Kazman el Método de Análisis de Desventajas de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Kazman propone el término enfoque arquitectónico dado que no todos los arquitectos están familiarizados con el lenguaje de estilos arquitectónicos, aún haciendo uso indirecto de estos. De cualquier forma, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros

EL SEI ha desarrollado ATAM a lo largo de varios años. El propósito de ATAM es: evaluar las consecuencias de las decisiones arquitectónicas en base en los atributos de calidad.

ATAM es un método en el que los principales puntos de análisis son:

- Descubrir riesgos: las alternativas que podrían crear problemas en el futuro en algunos atributos de calidad.
- Descubrir no riesgos: las decisiones que promueven cualidades que ayudan a cumplir los objetivos del negocio.
- Descubrir puntos de sensibilidad: alternativas donde un ligero cambio hace una diferencia significativa en algunos atributos de calidad.
- Descubrir desventajas - las decisiones que afectan a más de un atributo de calidad.

Realizar este análisis puede ser el objeto de las actividades de mitigación: por ejemplo, perfeccionar su diseño, un análisis más detallado, prototipo. Las desventajas pueden ser explícitamente identificadas y documentadas.

- Al realizar el análisis de ATAM traerá consigo una serie de beneficios como:
- Aclara cuáles son los atributos de calidad más importantes.
- Mejora la documentación de la arquitectura.

- Documenta las decisiones de la arquitectura base.
- Identifica riesgos en una fase temprana del ciclo de vida.
- Aumenta la comunicación entre los interesados.

El resultado final de ATAM es la mejora de las arquitecturas.

Las finalidades de ATAM es evaluar las consecuencias de las decisiones arquitectónicas a través de los atributos de calidad. ATAM es un proceso corto, facilita la interacción entre las múltiples partes interesadas, lo que lleva a la identificación de riesgos, las sensibilidades, y desventajas.

El propósito de ATAM NO es prever en el análisis, el objetivo es descubrir los riesgos creados por las decisiones arquitectónicas.

ATAM no funcionará si la Arquitectura de Software no ha sido creada aún, por lo que debe existir una arquitectura con un alcance definido y manejable, entre las precondiciones para aplicarlo se encuentran:

Los miembros del equipo de revisión analizarán los artefactos arquitectónicos y pueden ayudar a mejorar la documentación. El arquitecto debe preparar una arquitectura de presentación.

Los clientes deben preparar una presentación de los objetivos del negocio.

El equipo de evaluación revisará los artefactos de la arquitectura, presentaciones y leerá el material antes para familiarizarse con el dominio.

ATAM está estructurado en 4 fases y 9 pasos:

Presentación:

1- *Presentar ATAM.* El método describe a las partes interesadas (normalmente los representantes de los clientes, el arquitecto o el equipo de arquitectura, representantes de los usuarios, mantenedores, administradores, gerentes, probadores, integradores, etc.)

2- *Presentar los objetivos del negocio.* El líder del proyecto describe los objetivos que motivan el esfuerzo de desarrollo y, por tanto, lo que será la arquitectura primaria (por ejemplo, alta

disponibilidad, tiempo de salida al mercado o alta seguridad, requerimientos funcionales de alto nivel y requerimientos de atributos de calidad).

3- *Presentar la arquitectura.* El arquitecto presenta un panorama general de la arquitectura. Pueden ser las restricciones técnicas, tales como el sistema operativo, hardware, software o medios previstos para su uso; otros sistemas con los que el sistema debe interactuar; el estilo arquitectónico utilizado para hacer frente a los requisitos de los atributos de calidad. El equipo de evaluación comienza el análisis y la captura de los riesgos.

Investigación y Análisis:

4- *Identificar los enfoques arquitectónicos.* Comienzan a identificar los enfoques arquitectónicos que son fundamentales para la realización de los atributos de calidad objetivos.

5- *Generar atributos de calidad y el árbol de utilidad.* Identificar, priorizar y refinar los atributos de calidad más importantes, cuyo objetivo es la construcción de un árbol de utilidad.

- Un árbol de utilidad es un vehículo para el manejo de los requisitos de atributos de calidad específicos.
- Selecciona los objetivos de calidad para ser los nodos del alto nivel (por lo general, el rendimiento, modificabilidad, seguridad y disponibilidad)
- Los escenarios son las hojas del árbol de utilidad, anotado con los estímulos y las respuestas, y por orden de prioridad; estos deben ser lo más específicos posible.

La salida de un árbol de utilidad es la caracterización y priorización de los requisitos de atributos de calidad específicos.

La importancia para el éxito del sistema es Alta / Media / Baja; la dificultad para alcanzar el objetivo es Alta / Media / Baja a evaluación del arquitecto.

6- *Analizar los enfoques arquitectónicos.* El equipo de evaluación identifica los enfoques arquitectónicos desde el punto de vista de los atributos de calidad específicos para identificar los riesgos. Generar preguntas para los atributos de calidad de mayor prioridad (por ejemplo, un enfoque arquitectónico destinado a satisfacer objetivos de rendimiento será sometido a un análisis del rendimiento). Identificar “riesgos”, “no riesgos”, “puntos sensibles” y “desventajas”.

Pruebas

7- *Lluvia de ideas y dar prioridad a los escenarios.* Los interesados generan los escenarios apoyándose para facilitar el trabajo, en la realización de una lluvia de ideas, obteniendo un conjunto más amplio de los escenarios. A este conjunto de escenarios se le da prioridad a través de un proceso de votación de la totalidad de los participantes del grupo de interesados. Se agregan los nuevos escenarios al árbol de utilidad.

8- *Analizar los enfoques arquitectónicos.* Identificar los enfoques de arquitectura impactados por los escenarios generados en el paso anterior. Este paso continúa el análisis iniciado en el paso 6, usando los escenarios nuevos, y luego se documentan.

Presentación de informes

9- *Presentar los resultados.* Sobre la base de la información recogida en ATAM (estilos, escenarios, preguntas de atributos específicos, el árbol de utilidad, los riesgos, los puntos de sensibilidad, intercambios), el equipo evaluador presenta las conclusiones a las partes interesadas que se reunieron y potencialmente escribe un informe detallando esta información junto con cualquier propuesta de las estrategias de mitigación.

Académicamente, el momento de uso del ATAM es justo después de que la arquitectura se ha especificado cuando hay poco o ningún código. Sin embargo, en la práctica, ATAM ha sido muy efectivo en las siguientes situaciones:

- Evaluación de arquitecturas alternativas candidatas.
- Evaluación de los sistemas existentes antes de comprometerse a la mejora.
- Decidir entre actualizar o sustituir.

Revisión Activa de Diseños Parciales (ARID)

De acuerdo con Kazman el método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Revisión Activa del Diseño (ADR) y ATAM.

ADR es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la

documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

Kazman propone que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura.

Ante esta situación, y la necesidad de evaluación en las fases tempranas del diseño, Kazman proponen la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una Arquitectura de Software.

5.4 Estado de las Pruebas de Concepto.

Es indudable que el ambiente competitivo en el que se vive en el ámbito empresarial actualmente, requiere de promover los procesos y actividades de negocio que generan las ventajas competitivas de las empresas ante sus más fuertes competidores.

Por esto, desde hace ya varios años, se ha dado mayor importancia a las Tecnologías de la Información y su alineación con las estrategias del negocio para mejorar los procesos claves del negocio. Prueba de ello, es el incremento sustancial de adquisiciones de paquetes de software empresariales tales como el ERP (*Planificación de Recursos Empresariales*), con el cual los directivos de las empresas esperan tener integradas todas las áreas o departamentos de la empresa que apoyan para la generación de sus productos y servicios.

La calidad del software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia. La calidad es sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad.

La obtención de un software con calidad implica la utilización de metodologías o procedimientos estándares para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, en aras de lograr una mayor confiabilidad, mantenibilidad y facilidad de prueba, a la vez que eleven la productividad, tanto para la labor de desarrollo como para el control de la calidad del software.

Para garantizar la calidad de un software en una etapa temprana, un punto de partida importante es la Arquitectura de Software.

La Arquitectura de Software es la principal portadora de los atributos de calidad de un sistema, una pieza clave para el éxito de los proyectos de software; si la Arquitectura de Software es mal diseñada puede ser una fórmula para el desastre garantizado. Por tanto, para saber si una Arquitectura de Software es la adecuada para un sistema se debe llevar a cabo una evaluación de la misma.

Esta permite alcanzar la mayoría de los atributos de calidad esperados, es por ello que evaluar la Arquitectura de Software más que una necesidad constituye un reto. Esta evaluación puede realizarse mediante una serie de métodos que permiten, a través de la especificación de la calidad y la aplicación de técnicas variadas de evaluación, una estimación temprana de estos atributos. Existen diversos métodos; sin embargo, cada uno de éstos instancia la evaluación arquitectónica con rasgos y características particulares que lo hacen similar o diferente de los demás, aunque el propósito sigue siendo determinar la calidad.

La intención es obtener una arquitectura robusta, que cumpla con los servicios y la funcionalidad que espera el cliente, además de los atributos de calidad asociados que deben cumplirse. La necesidad de asegurar la calidad en las arquitecturas de software, ha provocado que varias de las organizaciones dedicadas a la producción de software hayan incluido en sus servicios la realización de pruebas de concepto a la arquitectura o prestar servicios a algún equipo de proyecto con este fin.

Entre las empresas a nivel mundial se pueden encontrar:

Compusof S.A[17].: Provee productos, servicios y soluciones orientadas al negocio, a las empresas que necesitan incorporar una tecnología adecuada a su sistema de trabajo. Al estar constituida sobre la base de la oferta integral de tecnologías de información, suministra de manera fiable hardware, software, servicios y soluciones avanzadas. Son integradores de los fabricantes líderes en el mercado tanto en hardware como en software y uno de los principales socios de Hewlett-Packard en España. Para la realización de pruebas de concepto definen las pruebas a realizar, implementan el entorno de prueba, realizan la instalación y prueba de aplicaciones, configuran Sistemas Operativos y aplicaciones y como resultado se obtiene la documentación de instalación, la configuración del Sistema Operativo y aplicaciones y por ultimo las recomendaciones adicionales.

*Hewlett-Packard Development Company, L.*P: ofrece servicios para aplicaciones empresariales como la solución “Empresa con latencia cero” (ZLE); donde, usando métodos ampliamente comprobados,

[17] **Compusof** proporciona desde hace más de dos décadas una oferta integral de servicios y soluciones en Tecnologías de la Información y Comunicaciones.

los profesionales de servicio de HP trabajan de cerca con el cliente para mapear la implementación de su tecnología ZLE de acuerdo a sus metas estratégicas. Desde la planeación final hasta la implementación final, dichos servicios abarcan todo el ciclo de vida del sistema asegurándose así de que su solución ZLE cubra las necesidades actuales y futuras de la empresa. Dentro de los servicios ZLE, HP incluye las Pruebas de concepto; que le permite construir un modelo funcional lo suficientemente detallado para determinar de manera específica el valor comercial y técnico de la solución ZLE para la empresa.

El SEI ha desarrollado los siguientes métodos para la evaluación de arquitecturas de sistema y software (explicados anteriormente):

- Método de Análisis de desventajas de la arquitectura (ATAM)
- Método de Análisis de Costos-Beneficios (CBAM)
- Revisiones Activas de Diseños Intermedios (ARID).

Estas técnicas pueden usarse solas o en combinación para obtener los beneficios de cualquier proyecto de desarrollo de software.

En Cuba, las instituciones dedicadas a la producción de software no tienen un proceso definido para la realización de pruebas de concepto, se desconocen las tecnologías que se emplean para su implementación, lo que provoca que no haya una cultura de realizar pruebas de concepto, por lo que las tomas de decisiones arquitectónicas pueden no ser las mejores para los sistemas a construir.

5.5 Herramientas para realizar las pruebas de concepto.

La utilización de herramientas para la realización de pruebas de concepto es muy importante y se recomienda para una mayor efectividad de dichas pruebas ya que permite determinar con mayor precisión y en menor tiempo muchos de los riesgos asociados a la arquitectura. Entre las principales herramientas que existen hoy día se encuentran:

- Máquina Virtual Java (MVJ).
- JMeter.
- pgBench.
- Administrador de tareas
- Herramienta de top en NU/Linux.

- Herramienta de Seguridad.

En el proceso de elaboración de pruebas el uso de herramientas trae consigo numerosas ventajas, entre ellas, mayor rapidez de ejecución, menor uso de recursos y el evitar pruebas obsoletas. Una herramienta de pruebas sin saber técnicas, prácticas y tener experiencia en pruebas es tan útil como un entorno de programación sin dominar el lenguaje. También es importante tener en cuenta el desarrollo de la tecnología, para usar herramientas que no estén obsoletas y se ajusten a las necesidades de una aplicación que puede estar desarrollada con una tecnología avanzada.

Conclusiones del Capítulo

En este capítulo se ha podido apreciar las diferentes pruebas que se le realizan a la Arquitectura de Software. Qué son las pruebas de concepto y las diferentes definiciones asociadas a estas, alcanzando un mayor dominio de los diferentes métodos que se utilizan para evaluar o probar la Arquitectura de Software.

Conclusiones Generales

La investigación desarrollada en la realización de este trabajo de diploma y el análisis de la documentación estudiada permitió el cumplimiento de los objetivos trazados. Luego del largo estudio realizado y las razones expuestas en el contenido, se logró establecer un marco teórico para la Arquitectura de Software en la actualidad. Se concluyó que para **construir** una Arquitectura de Software debe precisarse una serie de vistas que agrupen las definiciones necesarias para adquirir el conocimiento y las actividades requeridas para llevarla a cabo. Estas vistas son: Arquitectura de Sistema, Arquitectura de Datos, Arquitectura de Integración, Arquitectura de Presentación, Arquitectura de Seguridad y Arquitectura de Tecnología. Obteniéndose como resultado 6 vistas donde se identifican por cada una un promedio de 12 actividades, alrededor de 70 artefactos en total y un rol asociado a cada vista.

También para construir una arquitectura se necesita un equipo de trabajo y para concretar dicho equipo hay que enmarcarse en dos dimensiones, la Dimensión Corporativa y la Dimensión de Desarrollo. En la Corporativa hay un Arquitecto Corporativo, un Arquitecto de Procesos, un Arquitecto de Sistema y un Arquitecto de Tecnología, este último con un equipo compuesto por un Arquitecto de Seguridad, un Arquitecto Tecnológico y un Arquitecto de Presentación. En la Dimensión de Desarrollo se cuenta con un equipo de pruebas de concepto, un Arquitecto de Sistema al frente de un Arquitecto de Integración, un Arquitecto Temático que incluye un Arquitecto de Datos principal y otro para una línea en específico, además de un Arquitecto de Procesos, liderando ese equipo en la dimensión anterior.

Otro de los objetivos cumplimentados fue la estructuración de un expediente de proyecto para **organizar** la Arquitectura de Software, resultando una carpeta para cada una de las vistas propuestas, y otra carpeta donde se recogieron todos los temas referentes a la Gestión de la Calidad. Por último se finalizó que para **evaluar** la Arquitectura de Software se necesitan las pruebas de conceptos (PoC). Explicando los distintos tipos de PoC según lo que se quiera probar, identificándose además como principales técnicas de evaluación la Evaluación basada en escenarios, la Evaluación basada en simulación, la Evaluación basada en modelos matemáticos y la Evaluación basada en experiencia. En este tema también se expone que los principales métodos de evaluación para las PoC son el Método de Análisis de Arquitecturas de Software (SAAM), la Revisión Activa de Diseños Parciales o ARID y el Método de Análisis de Desventajas de Arquitectura (ATAM), todos propuestos por Kazman y con los que coinciden los autores de este trabajo.

El estudio del contenido de esta tesis y la implementación de las propuestas de los autores relacionadas con la Arquitectura del Software constituyen una guía de cómo organizar, construir y evaluar los aspectos tecnológicos y arquitectónicos en un proyecto de software.

Recomendaciones

Se recomienda, apoyado en el estudio realizado en este trabajo:

- Incluir elementos teóricos y metodológicos para la integración horizontal de las diferentes disciplinas especificadas en la propuesta de trabajo.
- Refinar las características metodológicas de los resultados presentados en el trabajo.
- Incluir propuesta práctica de elaboración de la evaluación y prueba de concepto de la Arquitectura de Software a partir de los referentes internacionales presentados en el trabajo.
- Se recomienda incorporar al estudio de esta disciplina, la Vista de Procesos y la Vista de Infraestructura con sus respectivos roles.
- Desarrollar temática Patrones de Integración.
- Desarrollar temática Política de reutilización y repositorio de componentes.
- Incorporar la disciplina de la Arquitectura de Software como parte del programa de las carreras asociadas a la rama de la informática en el país.
- Utilizar esta guía práctica para el establecimiento de una arquitectura en los proyectos productivos de software en cualquier organización, empresa o entidad.
- Emplear el resultado presentado en el trabajo para el aprendizaje y/o enseñanza de la disciplina de Arquitectura de Software.
- Continuar la investigación sobre el resultado obtenido y proponerlo como base referencial o material auxiliar en el desarrollo de futuros trabajos asociados al objeto de estudio de Arquitectura de Software.

Trabajos citados

1. **Reynoso, Carlos Billy.** willydev.ne. [En línea] http://www.willydev.net/descargas/prev/IntroArq.pdf.

2. **Rumbaugh, J.** *The Unified Modeling Language Reference.* s.l. : Addison Wesley, 1999.

3. **Bass, L., Clements, P., y Kazman, R.** *Software Architecture in Practice. .* s.l. : Addison Wesley., 1998.

4. **Garlan, D. y Perry, D. E.** *Special Issue on Software Architecture.* s.l. : IEEE Transctions on Software Engineering, 1995.

5. **Shaw, M.** *Some Patterns for Software Architecture.* s.l. : Addison-Wesley, 1996.

6. **Clements, Paul.** *A Survey of Architecture Description Languages.* Alemania : Proceedings of the International Workshop on Software Specification and Design, 1996.

7. **Christine Hofmeister, Robert Nord y Dilip Soni.** *Describing Software Architecture with UML. .* San Antonio : : En Proceedings of the First Working IFIP Conference on Software Architecture, IEEE Computer Society Press, , 1999.

8. **Booch, Grady.** *Object-Oriented Design with Applications.* s.l. : The Benjamin/Cummings Publishing Company, Inc, 1991.

9. **9126, Norma ISO.** mitecnologico. [En línea] http://www.mitecnologico.com/Main/LaNormaIsoIec9126.

10. **Bosert.** *ROI on Usability: A Business Perspective.* 1991 .

11. **Leder, Prassad.** *NIne management guidelines for the better cost estimating.* New York : ACM , 1992. ISSN:0001-0782.

12. **Ferren MacIntyre, Kenneth W. Estep, John McN. Sieburth.** *The cost of user-friendly programming: MacImage as example.* Rochester : Institute for Applied FORTH Research, Inc. , 1990. pág. 103. 0738-2022.

13. **Cooper, A.** *The Inmates Are Running the Asylum.* Indianapolis, Indiana : SAMS, Division of MacMillan Computer Publishing, 1999.

14. **IBM, Peter Eeles de.**

15. Bredemeyer Consulting. [En línea] http://www.bredemeyer.com..

16. **Vazquez, Ivis Rosa.** *El Rol del Arquitecto de Software.* 2008.

17. **Ricardo Baeza-Yates, Cuauhtémoc Rivera Loaiza,Javier Velasco Martín.** *Arquitectura de la información y usabilidad en la web.* 2004.

18. **Bichachi, Dra. Diana Susana.** *El uso de las Listas de Chequeo (Chesk-List) como herramienta para controlar la calidad de la ley.* s.l. : Instituto Internacional de Estudio y Formación sobre Gobierno y Sociedad (IIEFGS) Universidad del Salvador, 2009.

19. **Carriere, J., Kazman, R., Woods, S.** Toward a Discipline of ScenariobasedScenario-based Architectural Engineering. [En línea] [Citado el: 9 de mayo de 2002.] http://www.sei.cmu.edu/staff/rkazman/annals-scenario.pdf.

Bibliografía

1. **Bichachi, Dra. Diana Susana.** *El uso de las Listas de Chequeo (Chesk-List) como herramienta para controlar la calidad de la ley.* s.l. : Instituto Internacional de Estudio y Formación sobre Gobierno y Sociedad (IIEFGS) Universidad del Salvador, 2009.

2. Bredemeyer Consulting. [En línea] http://www.bredemeyer.com..

3. **IBM, Peter Eeles de.**

4. **Reynoso, Carlos Billy.** willydev.ne. [En línea] http://www.willydev.net/descargas/prev/IntroArq.pdf.

5. **Booch, Grady.** *Object-Oriented Design with Applications.* s.l. : The Benjamin/Cummings Publishing Company, Inc, 1991.

6. **Rumbaugh, J.** *The Unified Modeling Language Reference.* s.l. : Addison Wesley, 1999.

7. **Bass, L., Clements, P., y Kazman, R.** *Software Architecture in Practice. .* s.l. : Addison Wesley., 1998.

8. **Garlan, D. y Perry, D. E.** *Special Issue on Software Architecture.* s.l. : IEEE Transctions on Software Engineering, 1995.

9. **Shaw, M.** *Some Patterns for Software Architecture.* s.l. : Addison-Wesley, 1996.

10. **Clements, Paul.** *A Survey of Architecture Description Languages.* Alemania : Proceedings of the International Workshop on Software Specification and Design, 1996.

11. **Christine Hofmeister, Robert Nord y Dilip Soni.** *Describing Software Architecture with UML. .* San Antonio : : En Proceedings of the First Working IFIP Conference on Software Architecture, IEEE Computer Society Press, , 1999.

12. **Cooper, A.** *The Inmates Are Running the Asylum.* Indianapolis, Indiana : SAMS, Division of MacMillan Computer Publishing, 1999.

13. **Bosert.** *ROI on Usability: A Business Perspective.* 1991 .

14. **Leder, Prassad.** *NIne management guidelines for the better cost estimating.* New York : ACM , 1992. ISSN:0001-0782.

15. **Ferren MacIntyre, Kenneth W. Estep, John McN. Sieburth.** *The cost of user-friendly programming: MacImage as example.* Rochester : Institute for Applied FORTH Research, Inc. , 1990. pág. 103. 0738-2022.

16. **Vazquez, Ivis Rosa.** *El Rol del Arquitecto de Software.* 2008.

17. **Ricardo Baeza-Yates, Cuauhtémoc Rivera Loaiza,Javier Velasco Martín.** *Arquitectura de la información y usabilidad en la web.* 2004.

18. **9126, Norma ISO.** mitecnologico. [En línea] http://www.mitecnologico.com/Main/LaNormaIsoIec9126.

19. **Carriere, J., Kazman, R., Woods, S.** Toward a Discipline of ScenariobasedScenario-based Architectural Engineering. [En línea] [Citado el: 9 de mayo de 2002.] http://www.sei.cmu.edu/staff/rkazman/annals-scenario.pdf.

20. Introduccion a las Vistas. [En línea] http://www.dc.uba.ar/materias/arq-soft/2007/cuat1/descargas/.

21. *ISO/IEC. Software Engineering – Software quality – General overview, reference models and guide to Software Product Quality Requirements and Evaluation (SQuaRE).* s.l. : Reporte. , 2002. JTC1/SC7/WG6.

22. merinde. [En línea] http://merinde.rinde.gob.ve.

23. Software Engineering Institute(SEI). [En línea] http://www.sei.cmu.edu..

24. Windows Live. [En línea] 2009. [Citado el: 5 de enero de 2009.] http://diegumzone.spaces.live.com/blog/cns!1AD5096D63670065!290.

25. **Abrahamsson, Pekka.** [En línea] http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Abrahamsson:Pekka.html.

26. **Besnozov, Konstantin.** [En línea] abril de 1998. http://www.beznosov.net/konstantin/doc/cis6612-paper.pdf.

27. **Bustamante, Antonio Montes de Oca Sánchez de.** *Arquitectura de información y usabilidad: nociones básicas para los profesionales de la información. .* La Habana : s.n., 2004.

28. **Cano, Raúl de Villa.** *El Rol del Arquitecto de Software.* Medellín, España : s.n., 2008.

29. **Cano, Raúl de Villa.** *El Rol del Arquitecto de Software.* Medellin, España : s.n., 2008.

30. **Ciborra, C. U.** *Deconstructing the concept of strategic alignmen. .* s.l. : Scandinavian Journal of Information Systems., 1998.

31. **Fowler, Martin.** martinfowler. [En línea] 2001. [Citado el: 30 de septiembre de 2008.] http://www.martinfowler.com/articles/designDead.htm..

32. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal.** *Pattern-oriented software.*

33. **Headlines, IT.** *ActionCo Reinvents Business Leadership. .* 30 de Mayo de 2003.

34. **Highsmith, Jim.** *The great methodologies debate. Part I.* s.l. : Cutter IT Journal., 1991.

35. **Ing. Gustavo Andrés Brey, Ing. Gastón Escobar.** *Rol del Arquitecto de Software.* Buenos Aires, Argentina : s.n., 2005.

36. **Keen., P. G. W.** *Relevance anr rigor in information systems research.* s.l. : Elsevie, 1991.

37. **Kruchten, Philippe.** *The 4+1 View Model of Architecture.* s.l. : IEEE Software, 1995.

38. **Lasso, Adrián.** microsoft.com. [En línea] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art110.asp.

39. **Lerner, P.** [En línea] 2005. http://www.eitan.ws/publicaciones/ver_publicaciones.php.

40. **N. Dunlop, J. Indulska, K. A. Raymond.** *CORBA and RM-ODP: Parallel or divergent?, .* s.l. : Distributed Systems Engineering.

41. **Northrop, Paul Clements y Linda.** *Software architecture: An executive overview. .* s.l. : CMU/SEI-96-TR-003, ESC-TR-96-003., 1996.

42. **Pfleeger, Shari Lawrence.** *Ingeniería de Software: Teoría y Práctica.* Madrid : Prentice-Hall, 2002.

43. **Platt, Michael.** microsoft.com. [En línea] http://msdn.microsoft.com/architecture/overview/default.aspx?pull=/library/en-s/dnea/html/eaarchover.asp).

44. **Pressman, Roger.** *Ingeniería del Software: Un enfoque práctico. .* Madrid : McGraw Hill, 2001.

45. **Ronda León, R.** [En línea] 2005. http://www.nosolousabilidad.com/articulos/ai_cc_informacion.htm.

46. —. [En línea] 2008. http://www.nosolousabilidad.com/articulos/historia_arquitectura_informacion.htm.

47. **Ross, Douglas.** *Structured analysus (SA): A language for communicating ideas. .* s.l. : IEEE Transactions on Software Engineering., 1977.

48. **RUP.**

49. **Zachman., John A.** *A Framework for Information Systems Architecture.* s.l. : IBM Systems Journa, 1998.

50. **Nick Rozansk, Eóin Woods.** *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives (Hardcover).*

Glosario de Términos

Nota:

- **Glosario son términos con significado.**
- **Las siglas son ACRÓNIMOS**

ADLs: Architecture Description Language (Lenguajes de Descripción Arquitectónica). Lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos.

ARIS: Plataforma *ARIS* proporciona productos integrados de software que ayudan a las empresas a mejorar continuamente sus procesos de negocios.

Artefactos: Un artefacto es un producto tangible resultante del proceso de desarrollo de software.

Binding: Es una "ligadura" o referencia a otro símbolo más largo y complicado, y que se usa frecuentemente. Este otro símbolo puede ser un valor de cualquier tipo, numérico, de cadena, etc. o el nombre de una variable que contiene un valor o un conjunto de valores.

BPEL4WS: ***Business Process Execution Language***, o (*Lenguaje de Ejecución de Procesos de Negocio*). Es un lenguaje estandarizado por OASIS para la composición de servicios web. Básicamente, consiste en un lenguaje basado en XML diseñado para el control centralizado de la invocación de diferentes servicios Web, con cierta lógica de negocio añadida que ayuda a la programación en gran escala.

CEN: El **Comité Europeo de Normalización** (CEN), en francés *Comité Européen de Normalisation*, es una organización no lucrativa privada cuya misión es fomentar la economía europea en el negocio global, el bienestar de ciudadanos europeos y el medio ambiente proporcionando una infraestructura eficiente a las partes interesadas para el desarrollo, el mantenimiento y la distribución de sistemas estándares coherentes y de especificaciones.

CMM: Modelo de evaluación de los procesos de una organización. La visión general de los modelos basados en la madurez de las capacidades: Modelo de Capacidad y Madurez.

CMMI: Capability Maturity Model Integration (CMMI). Modelo para la mejora de procesos que proporciona a las organizaciones los elementos esenciales para procesos eficaces.

Las mejores prácticas CMMI se publican en los documentos llamados modelos. En la actualidad hay dos áreas de interés cubiertas por los modelos de CMMI: Desarrollo y Adquisición.

Independientemente de la constelación\modelo que opta una organización, las prácticas CMMI deben adaptarse a cada organización en función de sus objetivos de negocio.

Component Object Model (COM): Plataforma para componentes de software. Esta plataforma es utilizada para permitir la comunicación entre procesos y la creación dinámica de objetos, en cualquier lenguaje de programación que soporte dicha tecnología. Esencialmente COM es una manera de implementar objetos neutral con respecto al lenguaje, de manera que pueden ser usados en entornos distintos de aquel en que fueron creados.

Conectores: Servicios que solicita o brinda un componente o aplicación a otro componente o aplicación.

CORBA: En computación, **CORBA** (*Common Object Request Broker Architecture* — arquitectura común de intermediarios en peticiones a objetos), es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos. CORBA se puede considerar como un formato de documentación legible por la máquina, similar a un archivo de cabeceras, pero con más información.

CRC: La verificación de redundancia cíclica (abreviado, CRC) es un método de control de integridad de datos de fácil implementación. Es el principal método de detección de errores utilizado en las telecomunicaciones.

CRM: Modelo de gestión de toda la organización, basada en la orientación al cliente.

CVS: El Concurrent Versions System (CVS), también conocido como Concurrent Versioning System, es una aplicación informática que implementa un sistema de control de versiones que mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto y permite que distintos desarrolladores colaboren.

EJB (Enterprise JavaBeans): Los EJB proporcionan un modelo de componentes distribuidos estándar del lado del servidor. El objetivo de los EJB es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

ERPs (Enterprise resource planning): Los sistemas de planificación de recursos de la empresa son sistemas de gestión de información que integran y automatizan muchas de las prácticas de negocio asociadas con los aspectos operativos o productivos de una empresa. Los sistemas ERP son sistemas integrales de gestión para la empresa. Se caracterizan por estar compuestos por diferentes partes integradas en una única aplicación.

Frameworks: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y

un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Funcion Hash: Hash se refiere a una función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc.Un hash es el resultado de dicha función o algoritmo. Una función de hash es una función para resumir o identificar probabilísticamente un gran conjunto de información, dando como resultado un conjunto imagen finito generalmente menor (un subconjunto de los números naturales por ejemplo). Varían en los conjuntos de partida y de llegada y en cómo afectan a la salida similitudes o patrones de la entrada. Una propiedad fundamental del hashing es que si dos resultados de una misma función son diferentes, entonces las dos entradas que generaron dichos resultados también lo son.

HTML: **HyperText Markup Language** (*Lenguaje de Marcas de Hipertexto*). Lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

HTTP: **HyperText Transfer Protocol.** El protocolo de transferencia de hipertexto HTTP es el protocolo usado en cada transacción de la Web (WWW). HTTP fue desarrollado por el consorcio W3C y la IETF, colaboración que culminó en 1999. HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

IBM: **International Business Machines** o **IBM** es una empresa que fabrica y comercializa herramientas, programas y servicios relacionados con la informática. Tiene su sede en Arkmón (Estados Unidos) y está constituida como tal desde el 15 de junio de 1911, pero lleva operando desde 1888.

IBM, Microsoft, SEI, SUN, RUP y Bredemeyer: Principales compañías que llevan la delantera en lo referente a Arquitectura de Software.

Includes: Los #includes, server side includes (SSI) o en español, inclusiones de servidor, te permiten copiar el contenido de un fichero en otro fichero. La ventaja de las inclusiones es que te permite agrupar código en ficheros, y utilizarlo en todas las páginas que quieras.

IP: Una dirección IP es un número que identifica de manera lógica y jerárquica a una interfaz de un dispositivo (habitualmente una computadora) dentro de una red que utilice el protocolo IP (Internet Protocol) y que puede cambiar.

ISO, CEN, IEEE, OMG: Son los organismos encargados de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación. Su función principal es la de buscar la

estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional.

LDAP: (Lightweight Directory Access Protocol), (Protocolo Ligero de Acceso a Directorios) es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también es considerado una base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas. Habitualmente, almacena la información de autenticación (usuario y contraseña) y es utilizado para autenticarse aunque es posible almacenar otra información (datos de contacto del usuario, ubicación de diversos recursos de la red, permisos, certificados, etc.).

Licencia GNU GPL: Esta licencia GNU General Public License (GPL) se aplica en la mayoría de los programas realizado por la Free Software Foundation (FSF, Fundación del Software Libre) y en cualquier otro programa en los que los autores quieran aplicarla. También, muchos otros programas de la Free Software Foundation están cubiertos por la GNU Lesser General Public License (LGPL) e igualmente puede usarla para cubrir sus programas. Está diseñada para garantizar su libertad de compartir y modificar el software libre para garantizar la libertad de sus usuarios.

MDA: El **Monochrome Display Adapter** (**MDA**), también **tarjeta MDA** fue introducido en 1981. Fue de los primeros estándares de tarjetas de exhibición de vídeo para el computador IBM PC y los clones. El MDA no tenía modos gráficos, ofrecía solamente un solo modo de texto monocromático (el modo de vídeo 7), que podía exhibir 80 columnas por 25 líneas de caracteres de texto de alta resolución en un monitor TTL que mostraba la imagen en verde y negro.

Metis: Herramienta de modelado de procesos. Desarrollada por NCR y Computas, es una herramienta para la adquisición y visualización del conocimiento de la empresa, mayormente usada para la definición de la arquitectura empresarial, presenta además bondades para la organización, el análisis y el diseño.

Mínimo Privilegio: Es un principio que “requiere que a cada sujeto de un sistema se le otorgue el conjunto de privilegios más restrictivo (o la autorización más baja) necesario para el desempeño de sus tareas autorizadas. La aplicación de este principio limita el daño que puede generar un accidente, error o uso no autorizado”.

Multicast: Envío de la información en una red a múltiples destinos simultáneamente, usando la estrategia más eficiente para el envío de los mensajes sobre cada enlace de la red sólo una vez y creando copias cuando los enlaces en los destinos se dividen.

MVC: Es un patrón de Arquitectura de Software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la

página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

ODBC: **Open Database Connectivity** (**ODBC**). Estándar de acceso a Bases de datos desarrollado por Microsoft Corporation, el objetivo de *ODBC* es hacer posible el acceder a cualquier dato desde cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos almacene los datos.

Ofuscador de código: Programa que hace que aunque se tiene el código fuente, se enrevesado específicamente para ocultar su funcionalidad (hacerlo ininteligible). La ofuscación se refiere a encubrir el significado de una comunicación haciéndola más confusa y complicada de interpretar. Se refiere al acto deliberado de realizar un cambio no destructivo, ya sea en el código fuente de un programa informático o código máquina cuando el programa está en forma compilada o binaria, con el fin de que no sea fácil de entender o leer.

OMG: El **Object Management Group** u **OMG** (de sus siglas en inglés *Grupo de Gestión de Objetos*) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA. Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas. El grupo está formado por compañías y organizaciones de software como:

- Hewlett-Packard (HP)
- IBM
- Sun Microsystems
- Apple Computer.

OMT: Es una de las metodologías de análisis y diseño orientados a objetos, más maduros y eficientes que existen en la actualidad. La gran virtud que aporta esta metodología es su carácter de abierta (no propietaria), que le permite ser de dominio público y, en consecuencia, sobrevivir con enorme vitalidad. Esto facilita su evolución para acoplarse a todas las necesidades actuales y futuras de la ingeniería de software.

Parametrización: Propiedad de un módulo, o de una construcción sintáctica del lenguaje, para utilizar datos de varios tipos. Es un mecanismo muy útil porque permite aplicar el mismo algoritmo a tipos de datos diferentes; es una facilidad que permite separar los algoritmos de los tipos de datos, aumentando de esta manera la modularidad de los programas y minimizando la duplicación de código.

Patrones GRAPS: Patrones generales de software para asignación de responsabilidades, es el acrónimo de "General Responsibility Assignment Software Patterns". Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

Patrones de inyección de dependencia: Patrón de arquitectura orientado a objetos, en el que se inyectan objetos a una clase en lugar de ser la propia clase quien cree el objeto.

Peer-To-Peer: Se refiere a una red que no tiene clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores respecto de los demás nodos de la red. Es una forma legal de compartir archivos de forma similar a como se hace en el email o mensajeros instantáneos, sólo que de una forma más eficiente.

Plug and Play: Tecnología que permite a un dispositivo informático ser conectado a un ordenador sin tener que configurar (mediante un software específico (no controladores) proporcionado por el fabricante) ni proporcionar parámetros a sus controladores.

POS: Point of sale, punto de venta, designa la posibilidad de registrar transacciones comerciales en el lugar donde son efectuadas.

RAD: El RAD (Desarrollo rápido de aplicaciones o **Rapid Application Development**) es un proceso de desarrollo de software (en inglés, software development process), desarrollado inicialmente por James Martin en 1980. El método comprende el desarrollo iterativo, la construcción de prototipos y el uso de utilidades CASE (Computer Aided Software Engineering). Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

RDS: **Radio Data System** es una normalización que permite enviar datos inaudibles por la señal de una emisora de radio, normalmente de la FM, que se ven reflejados en la pantalla (display) del aparato de radio. Se utiliza en Europa y América Latina, aunque en Norteamérica usan uno muy similar, el RBDS (Radio Broadcast Data System).

RFP: **Request for Proposal,** Demanda de Propuestas, Llamada de Ofertas

RM-ODP: Modelo de Referencia para Procesamiento Distribuido Abierto

RUP: Rational Unified Process o Proceso Unificado Racional. Proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Scope: Alcance.

SEI: **Software Engineering Institute** (SEI). Instituto federal estadounidense de investigación y desarrollo, fundado por el Congreso de los Estados Unidos en 1984 para desarrollar modelos de evaluación y mejora en el desarrollo de software, que dieran respuesta a los problemas que generaba al ejército estadounidense la programación e integración de los sub-sistemas de software en la construcción de complejos sistemas militares. Financiado por el Departamento de Defensa de los Estados Unidos y administrado por la Universidad Carnegie Mellon.

Shell: Programa informático que actúa como interfaz de usuario para comunicar al usuario con el sistema operativo mediante una ventana que espera órdenes escritas por el usuario en el teclado (por ejemplo, PRINT CARTA.TXT), los interpreta y los entrega al sistema operativo para su ejecución. La respuesta del sistema operativo se muestra al usuario en la misma ventana. A continuación, el programa shell queda esperando más instrucciones.

Sistemas Distribuidos: Colección de computadores separados físicamente y conectados entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema.

SOAP: Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML, Es uno de los protocolos mas utilizados en los servicios Web.

SQL: Lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

Stack: Una **pila** (*stack* en inglés) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés *Last In First Out*, **último en *entrar*, primero en *salir***) que permite almacenar y recuperar datos. Se aplica en multitud de ocasiones en informática debido a su simplicidad y ordenación implícita en la propia estructura.

Stakeholders: Cualquier persona o entidad que es afectada por las actividades de una organización o empresa; por ejemplo, los trabajadores de esa organización, sus accionistas, las asociaciones de vecinos, sindicatos, organizaciones civiles y gubernamentales, etc.

SW-CMM: El **Modelo de Madurez de la Capacidad para el desarrollo de Software** (*Capability Maturity Model for Software*, **SW-CMM**) es un modelo de procesos para el desarrollo y mantenimiento de sistemas de software, diseñado sobre los criterios:

- La calidad de un producto o sistema es consecuencia directa de los procesos empleados en su desarrollo.

- Las organizaciones que desarrollan software presentan un atributo denominado madurez, cuya medida es proporcional a los niveles de capacidad e institucionalización de los procesos que emplean en su trabajo.

Tecnologías Legacy: Tecnología para la integración de protocolos. Ideado por IBM.

TOGAF: **The Open Group Architecture Framework** (TOGAF) (o **Framework Arquitectónico del Open Group**, en español) es un framework de Arquitectura Empresarial que proporciona un enfoque para el diseño, planeación, implementación y gobierno de una arquitectura empresarial de información. Esta arquitectura es modelada por lo general con cuatro niveles o dimensiones: Negocios, Tecnología (TI), Datos y Aplicaciones. Cuenta con un conjunto de arquitecturas base que buscan facilitarle al equipo de arquitectos definir el estado actual y futuro de la arquitectura.

Tunning: Es una serie de acciones que se toman para mejorar el rendimiento de la Base de Datos.

UML: Unified Modeling Language. Lenguaje Unificado de Modelado.

VACUUM: Realiza el mantenimiento de la base de datos, analiza y limpia una Base de datos.

Views and Beyond (V&B) Approach: La documentación de Arquitecturas de Software: Vista y más allá" (V & B) es un enfoque que sostiene que la documentación de una Arquitectura de Software es una cuestión de elegir un conjunto de opiniones pertinentes de la arquitectura, la documentación de cada uno de esos puntos de vista y, a continuación, la documentación de la información que se aplica a más de una opinión o para el conjunto de puntos de vista en su conjunto. Detalles de este enfoque incluyen un método para elegir los más relevantes puntos de vista, las plantillas estándar para documentar los puntos de vista y la información más allá de ellas, y las definiciones de las plantillas de contenido.

Visual Paradigm: Herramienta de modelado UML y que cuenta con una versión gratuita denominada Community Edition.

Webmaster: La palabra **Webmaster** es un término comúnmente usado para referirse a las personas responsables de un sitio web específico. En una página pequeña, el Webmaster será típicamente el dueño, diseñador, desarrollador y programador, además de actuar como el encargado en la redacción, edición y publicación del contenido. En sitios más grandes, las funciones del Webmaster serán mayores, actuando este como coordinador y supervisor de las actividades de todos los integrantes que colaboran con él. En ocasiones es también un empleado del dueño del sitio web.

Wrapper: Término que se refiere a una clase Java en la programación orientada a objetos

WS-I: **Web Services Interoperability.** Su objetivo es fomentar y promover la Interoperabilidad de Servicios Web (**WS-I**) sobre cualquier plataforma, sobre aplicaciones, y sobre lenguajes de programación. Su intención es ser un integrador de estándares para ayudar al avance de los

servicios web de una manera estructurada y coherente. La WS-I ha organizado los estándares que afectan a la interoperabilidad de los servicios web en una *pila* basada en funcionalidades.

XML: *E**x**tensible **M**arkup **L**anguage* (Lenguaje de Marcas). Metalenguaje extensible de etiquetas, no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.