

CUADERNO DE APUNTES

TALLER DE ANÁLISIS DE SISTEMAS



Estimado Estudiante de AIEP, en este Cuaderno de Apuntes encontrarás conceptos, ideas centrales y aplicaciones que reforzarán los aprendizajes esperados a desarrollar en este módulo.

Esperamos que el contenido y las actividades de este cuaderno te sean de utilidad y te orienten en tu proceso formativo.

Mucho Éxito.

Dirección de Desarrollo Curricular y Evaluación

VICERRECTORÍA ACADÉMICA AIEP.

I UNIDAD DE APRENDIZAJE: ANÁLISIS DE SISTEMAS

Aprendizaje Esperado 1

Determinan alcance de análisis de sistemas, considerando diseño de sistemas, tipos de sistemas, roles de un analista y fundamentos asociados.

1.1. Conceptos en diseño de sistemas

La palabra Diseño, denota trazo, delineación o descripción. Comprende y combina características o detalles que a menudo requieren la preparación de dibujos o planes preliminares. En otras palabras, el diseño de sistemas es una combinación de actividades y procedimientos para lograr aquellos objetivos organizacionales.

Los pasos que se deben seguir cuando se diseña un nuevo sistema son:

1. Examine todos los datos posibles (entradas y salidas de información)
2. Concéntrese y piense en forma creativa
3. Proporcione diferentes entradas, salidas, operaciones, controles y técnicas de procedimiento.
4. Primero evalúe los procedimientos más importantes (priorizar por necesidad y valor para el negocio)
5. Examine diversas alternativas (según presupuesto y factibilidades técnicas)

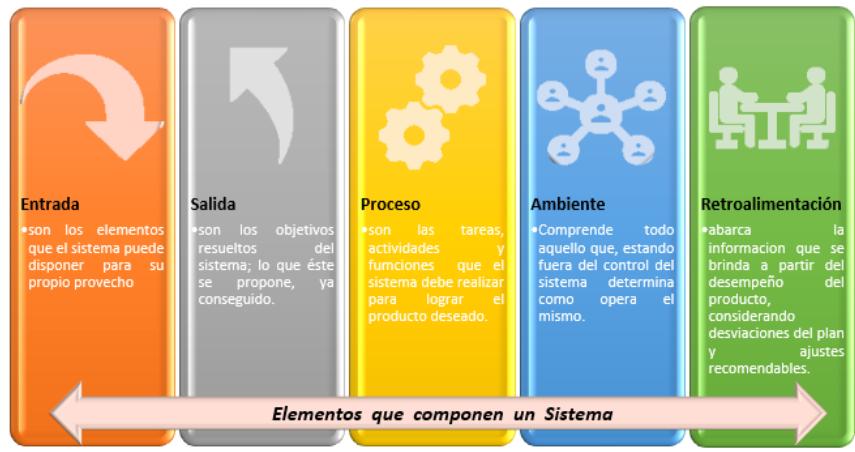


Figura N° 1: Elementos de un Sistema
 Fuente: Elaboración Propia

1.2. Tipos de sistemas de software

- **Sistema de Procesamiento de Transacciones (TPS):** Son aquellos sistemas que procesan grandes cantidades de datos, los cuales están relacionados con las transacciones de los negocios, como, por ejemplo: Las nóminas y los inventarios.

- **Sistema de Procesamiento de Datos (OAS)¹:** Son aquellos sistemas de información que procesan grandes volúmenes de información generados en las funciones administrativas, como el control de inventarios. Se realizan de manera automática, es decir, ejecutan las actividades diariamente.
- **Sistemas para la Administración (MIS):** Son sistemas que se basan en los datos obtenidos, el procesamiento de datos, y requieren la participación del ser humano, software, y hardware. Estos sistemas usan datos para el análisis y la posterior toma de decisiones.
- **Sistema de apoyo a la toma de decisiones (DSS):** Enfatizan cada etapa de la toma de decisiones gracias a la información obtenida.
- **Sistemas expertos e inteligencia artificial (AI):** Es el campo principal de los sistemas expertos que permite desarrollar máquinas que cuenten con un desempeño inteligente. Tiene dos áreas que son: La comprensión del lenguaje natural y la habilidad para interiorizar los problemas hasta alcanzar una conclusión lógica.
- **Sistemas de soporte de decisiones en grupo y sistemas de trabajo colaborativo asistido por computadora (GDSS)²:** Las organizaciones confían cada vez más en los equipos para tomar decisiones en conjunto. Estos sistemas se utilizan en cuartos especiales equipados con varias configuraciones, permiten a los miembros de los grupos interactuar con el soporte electrónico (a menudo en la forma de software especializado) y un facilitador de grupo especial. El objetivo es lograr que resuelvan un problema con la ayuda de varios apoyos como encuestas, cuestionarios, lluvia de ideas y creación de escenarios. Algunas veces, los sistemas GDSS se consideran bajo el término más general de sistemas de trabajo colaborativo asistido por computadora (CSCWS), que podría incluir el soporte de software conocido como groupware para colaborar en equipo mediante computadores conectados en red. Los sistemas de soporte de decisiones en grupo también se pueden utilizar en un ambiente virtual.
- **Sistemas de soporte para ejecutivos (ESS)³:** Los ESS ayudan a los ejecutivos a organizar sus interacciones con el entorno externo ofreciendo tecnologías de gráficos y comunicaciones en sitios accesibles como salas de juntas u oficinas corporativas personales. Aunque los sistemas ESS se basan en la información que generan los sistemas TPS y MIS, ayudan a sus usuarios a enfrentar los problemas relacionados con decisiones no estructuradas inespecíficas de una aplicación, para lo cual crean un entorno que les ayude a pensar sobre los problemas estratégicos de una manera informada. Los sistemas ESS extienden las capacidades de los ejecutivos y les ofrecen soporte para que puedan entender mejor sus entornos.

¹ Análisis y Diseño de Sistemas 8º edición Kendal y Kendal, capítulo 1, “Sistemas, Roles y metodologías de desarrollo” Parte I Fundamentos del Análisis de Sistemas.

² Kendall. (2011). análisis y diseño de sistemas - 8b: edición. Pearson Educación.

³ Tirado, L. F. F. (2004, June 15). Sistemas de información para ejecutivos y toma de decisiones. Gestiopolis.com. <https://www.gestiopolis.com/sistemas-informacion-ejecutivos-toma-decisiones/>

El diseño de un sistema de información se define como el proceso de aplicar ciertas técnicas y principios con el propósito de definir un dispositivo, un proceso o un sistema, con suficientes detalles como para permitir su interpretación y realización física.

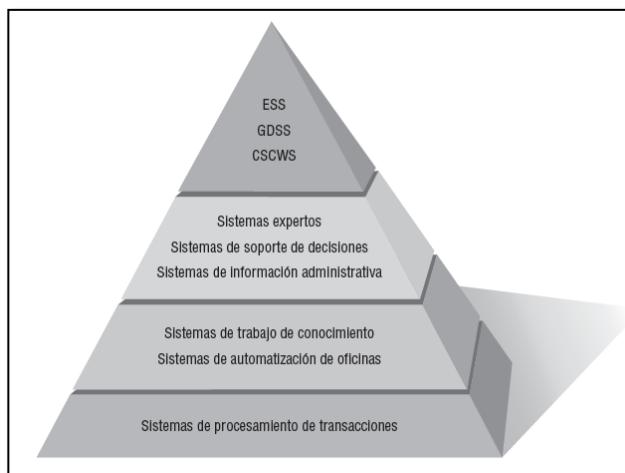


Figura N° 2: Tipos de Sistemas de Software
 (Kendall & Kendall, 2011)

1.3. Fundamentos del análisis de sistemas

El análisis de sistemas es el estudio de un sistema y sus componentes. Es un requerimiento previo para el diseño de sistemas, la especificación de un sistema nuevo y mejorado. El análisis de sistemas es un término que en forma colectiva describe las fases iniciales del desarrollo de sistemas. Nunca ha habido una definición universalmente aceptada del análisis de sistemas. De hecho, nunca ha habido un acuerdo universal acerca de cuándo termina un análisis de sistemas de información y cuándo comienza el diseño de sistemas de información⁴.

Otra definición de análisis de sistema es una técnica de solución de problemas que descompone el sistema en sus componentes para estudiar el grado en que estos funcionan e interactúan para lograr su propósito.

Las fases de desarrollo de un proyecto de desarrollo de sistemas de información se centran en los problemas y requerimientos de negocios, con independencia de la tecnología que pueda usarse o se use para implementar una solución al problema.

- Fase 1: Definición de alcance
- Fase 2: Análisis de problemas
- Fase 3: Análisis de requerimientos

⁴ Kendall. (2011). análisis y diseño de sistemas - 8b: edición. Pearson Educación.

- Fase 4: Diseño lógico
- Fase 5: Análisis de decisión

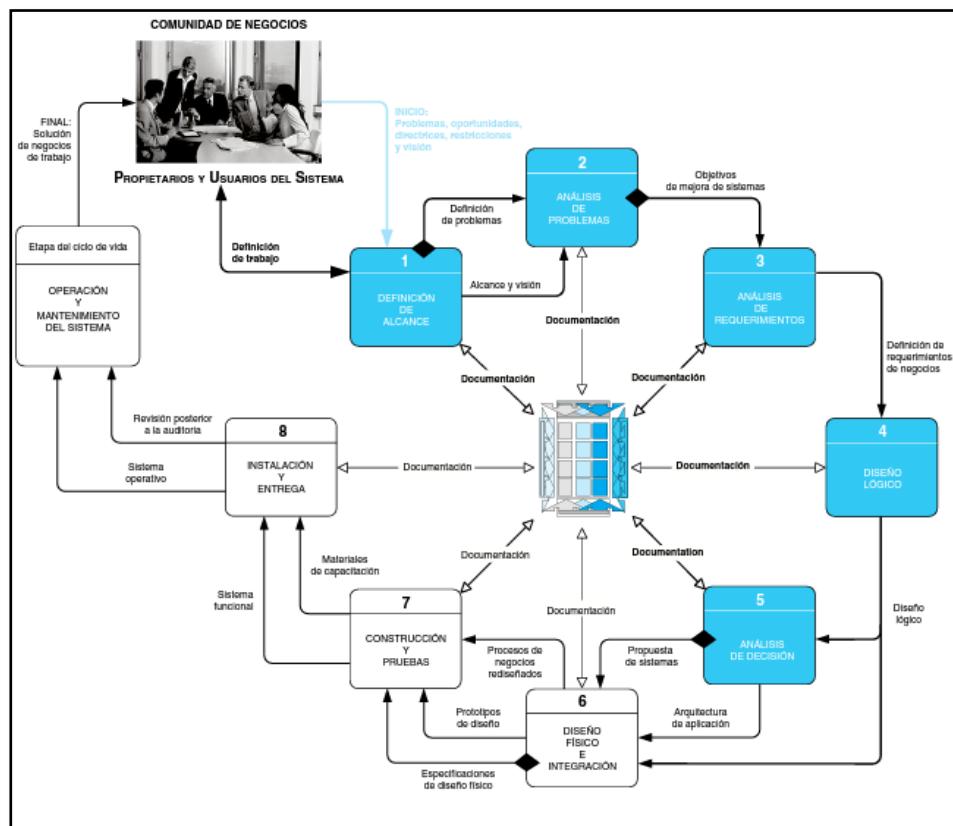


Figura N° 3: Contexto del Análisis de Sistemas
 (Bentley & Whitten, 2008)

1.4. Roles de un analista en el desarrollo de software

El especialista en análisis y diseño es conocido como analista y evalúa en forma sistemática cómo interactúan los usuarios con la tecnología y cómo operan las empresas, para lo cual examina los procesos de entrada/salida de los datos y la producción de información con la intención de mejorar los procesos organizacionales.

El analista debe ser capaz de trabajar con todo tipo de personas y debe tener experiencia con computadores. El analista desempeña muchos roles y algunas veces tiene que lidiar con varios al mismo tiempo. Los tres principales roles del analista de sistemas son como consultor, experto de soporte y agente de cambios.

- **Analista de sistemas como consultor:** Con frecuencia el analista actúa como consultor para las personas y sus empresas para lidiar con temas relacionados con los sistemas de información dentro de la empresa. Este rol puede ser ventajoso ya que al ser consultor externo tiene una perspectiva distinta de la empresa a diferencia de sus trabajadores.

También implica que los analistas externos están en desventaja, ya que al mirar de afuera la empresa no podrán conocer la verdadera cultura organizacional⁵.

- **Analista de sistemas como experto de soporte:** En este rol el analista se basa en su experiencia profesional sobre hardware y software y su uso en los negocios. A menudo este trabajo no es un verdadero proyecto de sistemas, si no que supone una pequeña modificación o decisión que afecta a un solo departamento.
- **Analista de sistemas como agente de cambio:** Es el rol más extenso y responsable del analista de sistema, ya sea interno o externo, para la empresa. Podemos definir a un agente de cambio como una persona que actúa como catalizador para el cambio, desarrolla un plan de cambio y trabaja con otros para facilitarlo.

1.5. Proyectos de desarrollo de software

¿Qué es un proyecto?

Es un emprendimiento temporario para crear un producto o servicio único. Es temporario ya que termina cuando se cumplen los objetivos; y es único porque se definen progresivamente sus características (PMBOK, 2013⁶).

Los proyectos tienen parte interesada (stakeholders): Involucrados (participan); interesados afectados positiva o negativamente. Algunos son: Gerente del proyecto, usuarios, cliente, organización ejecutora, patrocinador, etc.

Para cada proyecto se establecen el alcance del producto (final), sus características; y del proyecto y trabajos que se incluyen (y los que no).

1.5.1. Principales características de un proyecto⁷

Todos los tipos de proyectos tienen en común una serie de características:

- Cuentan con un propósito
- Se resumen en objetivos y metas
- Cuentan con un alcance
- Se han de ajustar a un plazo de tiempo limitado
- Cuentan con, al menos, una fase de planificación, una de ejecución y una de entrega
- Se orientan a la consecución de un resultado

⁵ Análisis y Diseño de Sistemas 8° edición Kendal y Kendal, capítulo 1, “Sistemas, Roles y metodologías de desarrollo” Parte I Fundamentos del Análisis de Sistemas.

⁶ https://sistematic.files.wordpress.com/2017/07/guia_de_los_fundamentos_para_la_direccion_de_proyectos-pmbok_5ta_edicion_espanol.pdf

⁷ Tipos de proyectos y sus principales características. (n.d.). Obsbusiness.School. Retrieved March 18, 2021, from <https://www.obsbusiness.school/blog/tipos-de-proyectos-y-sus-principales-caracteristica>

- Involucran a personas, que actúan en base a distintos roles y responsabilidades
- Se ven afectados por la incertidumbre
- Han de sujetarse a un seguimiento y monitorización para garantizar que el resultado es el esperado
- Cada uno es diferente, incluso de los de similares características

1.5.2. Tipos de proyecto

Existen muchos tipos de proyecto y por ello es habitual que un equipo de proyecto a menudo incluya a personas que normalmente no trabajan juntas, por proceder de organizaciones distintas o por provenir de ubicaciones geográficas diferentes. Los más comunes se indican en la tabla (Figura N° 4).⁸

Criterio	Tipo
Grado de dificultad para su ejecución	Simples Complejos
Procedencia del Capital	Públicos Privados Mixtos
Grado experimentación del proyecto y sus objetivos	Experimentales Normalizados
Sector productivo	Construcción - Energía Minería - Transformación Medio ambiente - Industriales - Servicios
Ámbito	Ingeniería - Económicos Fiscales - Legales Médicos -Matemáticos Artísticos -Literarios Tecnológicos - Informáticos
Orientación	Productivos - Educativos Sociales - Comunitarios Investigación
Área de influencia	Internacionales - Nacionales Regionales - Locales

Figura N° 4: Tipos de Proyectos
 (Abarca, 2019)

- **Gestión de Proyectos:** Aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades de un proyecto, para cubrir o superar las necesidades y expectativas para un proyecto.

⁸ Marvo Villalobo Abarca. Informe Proyectos de desarrollo de Software, versión 1.0, Universidad de Tarapacá, Arica -Chile.
 Departamento Ingeniería en computación e informática

- **Procesos de un proyecto:** Los procesos asociados al desarrollo de un proyecto son:
 - **Procesos de gestión del proyecto:** Refieren a describir y organizar el trabajo del proyecto.
 - **Procesos orientados al producto:** Refieren a especificar y crear el producto del proyecto definidos normalmente por el ciclo de vida del proyecto y varían por área de aplicación.

Ambos grupos de procesos interactúan a lo largo del proyecto: Por ejemplo, es imposible definir el alcance del producto sin conocimiento del área de aplicación relacionada con cómo construir el producto.

En la Figura N° 5, se muestran los cinco procesos de gestión para un proyecto. Inicio, Planificación, Ejecución, Control y Cierre.

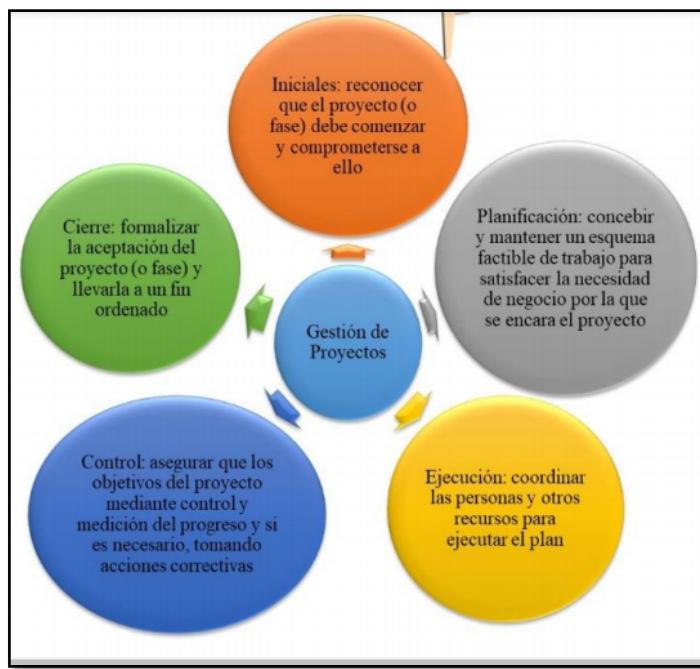


Figura N° 5: Procesos de Gestión para un Proyecto
(Villalobos Abarca, 2019)

1.6. Alcance del análisis de sistemas

Es un conjunto de procedimientos o programas relacionados de manera que juntos forman una sola unida; Esto se lleva a cabo teniendo en cuenta ciertos principios:

- Debe presentarse y entenderse el dominio de la información de un problema
- Defina las funciones que debe realizar el Software
- Represente el comportamiento del software a consecuencias de acontecimientos externos

- Divida en forma jerárquica los modelos que representan la información, funciones y comportamiento

El proceso debe partir desde la información esencial hasta el detalle de la Implementación. La función del Análisis puede ser dar soporte a las actividades de un negocio, o desarrollar un producto que pueda venderse para generar beneficios.

Un Análisis de Sistema se lleva a cabo teniendo en cuenta los siguientes objetivos en mente:

- Identifique las necesidades del Cliente
- Evalúe qué conceptos tiene el cliente del sistema para establecer su viabilidad
- Realice un Análisis Técnico y Económico
- Asigne funciones al Hardware, Software, personal, base de datos, y otros elementos del Sistema
- Establezca las restricciones de presupuestos y planificación temporal
- Cree una definición del sistema que forme el fundamento de todo el trabajo de Ingeniería

Para lograr estos objetivos se requiere tener un gran conocimiento y dominio del Hardware y el Software, así como de la Ingeniería Humana (Manejo y Administración de personal), y administración de base de datos.

1.6.1. Objetivos del análisis de sistema de información

- **Identificación de Necesidades:** Es el primer paso del análisis del sistema, en este proceso el Analista se reúne con el cliente y/o usuario, e identifican las metas globales, se analizan las perspectivas del cliente, sus necesidades y requerimientos, sobre la planificación temporal y presupuestal, líneas de mercadeo y otros puntos que puedan ayudar a la identificación y desarrollo del proyecto.

Antes de su reunión con el analista, el cliente prepara un documento conceptual del proyecto, aunque es recomendable que este se elabore durante la comunicación Cliente – Analista, ya que de hacerlo el cliente solo, de todas maneras, tendría que ser modificado, durante la identificación de las necesidades.

- **Estudio de Viabilidad:** Muchas veces, cuando se emprende el desarrollo de un proyecto de Sistemas, los recursos y el tiempo no son realistas para su materialización sin tener pérdidas económicas y frustración profesional. La viabilidad y el análisis de riesgos están relacionados de muchas maneras, si el riesgo del proyecto es alto, la viabilidad de producir software de calidad se reduce, sin embargo, se deben tomar en cuenta cuatro áreas principales de interés.⁹

⁹ edukativos. (2016, May 7). Identificación de necesidades en el desarrollo del software. Edukativos.Com. <https://edukativos.com/apuntes/archives/10543>

- **Viabilidad económica:** Una evaluación de los costos de desarrollo, comparados con los ingresos netos o beneficios obtenidos del producto o Sistema desarrollado.
- **Viabilidad Técnica:** Un estudio de funciones, rendimiento y restricciones que puedan afectar la realización de un sistema aceptable.
- **Viabilidad Legal:** Es determinar cualquier posibilidad de infracción, violación o responsabilidad legal en que se podría incurrir al desarrollar el Sistema.
- **Alternativas:** Una evaluación de los enfoques alternativos del desarrollo del producto o Sistema.
- **Análisis económico y técnico de los sistemas de información:**
 - **El análisis económico** incluye lo que llamamos, el análisis de costos/beneficios, significa una valoración de la inversión económica comparado con los beneficios que se obtendrán en la comercialización y utilidad del producto o sistema.

Muchas veces, en el desarrollo de Sistemas de Computación, estos son intangibles y resulta un poco dificultoso evaluarlo, esto varía de acuerdo con las características del Sistema. El análisis de costos/beneficios es una fase muy importante porque de ella depende la posibilidad de desarrollo del Proyecto.
 - **En el Análisis Técnico,** el Analista evalúa los principios técnicos del Sistema y, al mismo tiempo recoge, información adicional sobre el rendimiento, fiabilidad, características de mantenimiento y productividad.

Los resultados obtenidos del análisis técnico son la base para determinar si continuar o abandonar el proyecto, si hay riesgos de que no funcione, no tenga el rendimiento deseado, o si las piezas no encajan perfectamente unas con otras.
- **Modelado de la arquitectura del Sistema:** Cuando queremos dar a entender mejor lo que vamos a construir en el caso de edificios, herramientas, aviones, máquinas, se crea un modelo idéntico, pero en menor escala. Sin embargo, cuando aquello que construiremos es un Software, nuestro modelo debe tomar una forma diferente, deben representar todas las funciones y subfunciones del Sistema. Los modelos se concentran en lo que debe hacer el sistema no en como lo hace, pueden incluir notación gráfica, información y comportamiento del Sistema. Todos los Sistemas basados en computadores pueden modelarse como transformación de la información empleando una arquitectura del tipo entrada y salida.
- **Especificaciones del Sistema:** Es un Documento que sirve como fundamento para la Ingeniería Hardware, software, Base de datos, e ingeniería Humana. Describe la función y

rendimiento de un Sistema basado en computadores y las dificultades que estarán presentes durante su desarrollo. Las Especificaciones de los requisitos del software se producen en la terminación de la tarea del análisis.

ACTIVIDAD DEL APRENDIZAJE ESPERADO N° 1

Actividad N° 1

- I. Indique a qué tipo de sistemas pertenecen los siguientes Software; Marque con una (X) según corresponda.

Tipos de Software	Tipos de Sistemas						
	TPS	OAS	MIS	DSS	AI	GDSS	ESS
Sistema de Ventas y Marketing							
Sistema de Base de datos (Access)							
Sistema de Control de Inventario							
Sistema de pronóstico de presupuesto a cinco años							
sistema de apoyo clínico que podría identificar el cáncer en etapas tempranas.							
Sistemas de Manufactura y Producción							
Sistema de Análisis de fijación de precios y rentabilidad							
Sistemas de Finanza y Contabilidad							
Software de diseño y dibujo industrial (AutoCAD)							
Sistemas de Recursos Humanos							
Sistema de para el Análisis de Reubicación de Personal							
Sistema de Reserva de asientos para una línea aérea							
Sistema de Pronostico de tendencia de ventas en periodos anuales							
Sistema de planificación de recursos (Gantt Project)							
Sistema de Programación de la Producción							
Sistema de Votación Online							
Sistema para el diagnóstico de infecciones bacterianas.							

Aprendizaje Esperado 2

Determinan metodología de desarrollo para proyecto de software, considerando etapas del ciclo de vida de software, problemáticas asociadas y metodologías ágiles.

2.1. Ciclo de vida del desarrollo de sistemas: Etapas de planificación, análisis, diseño, implementación, testing y mantención

El ciclo de vida es un proceso que provee una solución para el desarrollo de un sistema el cual identifica etapas y secuencias en el desarrollo, encapsula el conocimiento de casos pasados y facilita el desarrollo de nuevos casos. El ciclo de vida contiene etapas que incluyen la identificación de requerimientos, diseño, implementación, testeo, puesta en marcha, operación y mantenimiento, las cuales se pueden englobar en dos etapas: 1) desarrollo de sistemas y 2) operación y mantenimiento de sistemas; es decir “primero se construye y, luego se usa y se mantiene”.

Tarde o temprano el ciclo culmina con el desarrollo de un nuevo sistema.

Estas etapas son un reflejo del proceso que se sigue a la hora de resolver cualquier tipo de problema que básicamente requiere:

- Comprender el problema (análisis)
- Plantear una posible solución, considerando soluciones alternativas (diseño)
- Llevar a cabo la solución planteada (implementación)
- Comprobar que el resultado obtenido es correcto (pruebas)

Las etapas adicionales de planificación, instalación y mantenimiento que aparecen en el ciclo de vida de un sistema de información son necesarias en el mundo real porque el desarrollo de un sistema de información conlleva costos asociados, por lo que es necesaria la planificación ya que una vez construido el sistema este debe utilizarse con el fin de recuperar la inversión de lo contrario no tendría sentido haber invertido en su desarrollo.

- **Etapa 1 planificación:** En esta etapa se deben realizar las siguientes tareas iniciales del proyecto que incluyen actividades tales como la determinación del ámbito del proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados al proyecto, una estimación del costo del proyecto, su planificación temporal y la asignación de recursos a las distintas etapas del proyecto.

- **Delimitación del ámbito del proyecto:** Es esencial establecer de antemano qué asuntos han de resolverse durante la realización del proyecto y cuáles se dejarán fuera. Estos últimos han de indicarse explícitamente si es necesario. Como resultado se obtiene un documento breve de 1 o 2 páginas en el que se describe el problema que nuestro sistema pretende resolver. Este documento es considerado de alto nivel de funcionalidad y debe formar parte del contrato que se firme con el cliente.
 - **Estudios de viabilidad:** Este documento sirve para identificar los principales factores que hacen fracasar proyectos de desarrollo de software y los ingredientes claves que pueden ayudar a

reducir el índice de fracasos. Antes de comenzar un proyecto se debe evaluar la viabilidad económica, técnica y legal del mismo, las cuales deben ajustarse a la realidad actual.

- **Análisis de riesgos:** La evaluación de riesgos se utiliza para identificar riesgos que puedan afectar negativamente el plan de nuestro proyecto, estimar la probabilidad de que el riesgo se materialice y analizar su posible impacto en nuestro proyecto. Algunos riesgos pueden ser, que un miembro clave de nuestro equipo abandone el proyecto, modificar erróneamente o eliminar parte del código o también si falla algún servicio de hardware. Pueden ser algunos de los riesgos que se hacen presentes en el desarrollo de un software.

Una vez analizados los riesgos potencialmente más peligrosos podemos recurrir a distintas técnicas de control de riesgos como, por ejemplo, elaborar planes de contingencia para eliminar de raíz o mitigar el riesgo o también añadir cierta holgura a la planificación de nuestro proyecto.

- **Estimación de costos:** Una de las tareas más complejas de cualquier proyecto de desarrollo de software es la estimación inicial de costos de algo que aún no se conoce, la incertidumbre en la estimación es inevitable, pero en ocasiones puede reducirse. Cuantos más datos históricos recopilemos y más precisa sea la información de la que dispongamos acerca de nuestro proyecto, mejor será nuestra estimación.

Resulta aconsejable utilizar varias técnicas de estimación y contrastar los resultados con ellas obtenidos. Por ejemplo, podemos realizar una estimación en función del costo de un proyecto similar, utilizar algún modelo matemático de estimación como por ejemplo [COCOMO](#) o similar y realizar una tercera estimación descomponiendo nuestro proyecto en tareas. Si los resultados obtenidos con las distintas estimaciones son similares, nuestra estimación será buena.

- **Planificación temporal y asignación de recursos:** Una planificación excesivamente detallada puede resultar contraproducente. Cualquier error de planificación causado por algún imprevisto nos forzará a replanificar el resto del proyecto, retrasando aún más nuestro proyecto. Una planificación por semanas suele ser razonable para afrontar con comodidad las contingencias con las que nos vayamos encontrando sin tener que estar continuamente reajustando el plan del proyecto.

Pase lo que pase, la planificación del proyecto ha de reajustarse cada vez que cambian las circunstancias de este. La planificación es fundamental en la gestión de un proyecto de desarrollo de software. Un plan que no se ajusta a la realidad no sirve de mucho.

- **Etapa 2 Análisis:** Lo primero que debemos hacer para construir un sistema es averiguar qué es exactamente lo que tiene que hacer el sistema. La etapa de análisis corresponde al proceso mediante el cual se intenta descubrir qué es lo que realmente se necesita y se llega a una comprensión adecuada de los requerimientos del sistema. El problema es que, de primeras, puede que ni nuestro cliente sepa qué es exactamente lo que necesita. Por tanto, deberemos ayudarle a averiguarlo con ayuda de distintas técnicas que más adelante aprenderemos a utilizar. Porque el costo de construir correctamente un sistema a la primera es mucho menor que el costo de construir un sistema que habrá que modificar más adelante. Cuanto antes se detecta un error, mejor.

Se estima que un 25% de los requerimientos iniciales de un sistema cambian antes de que el sistema comience a utilizarse. Muchas prácticas resultan efectivas para gestionar adecuadamente los requerimientos de un sistema y, en cierto modo, controlar su evolución. Un buen analista debería tener una formación adecuada en:

- **Técnicas de elicitation de requerimientos**
 - **Herramientas de modelado de sistemas**
 - **Metodologías de análisis de requerimientos**
- **Etapa 3 diseño:** Mientras que los modelos utilizados en la etapa de análisis representan los requisitos del usuario desde distintos puntos de vista, los modelos que se utilizan en la fase de diseño representan las características del sistema que nos permitirán implementarlo de forma efectiva.

Un software bien diseñado debe exhibir determinadas características tomadas con los requerimientos del sistema. En la fase de diseño se han de estudiar posibles alternativas de implementación y se ha de decidir la estructura general que tendrá el sistema. Esto se llama diseño arquitectónico. Igual en la etapa de análisis creábamos distintos modelos en función del aspecto del sistema en que centrábamos nuestra atención, el diseño de un sistema de información también presenta distintas facetas, por un lado, es necesario el diseño de la base de datos y, por otro lado, hay que diseñar las aplicaciones que permitirán al usuario el uso del sistema, las cuales son llamadas interfaz de usuario.

- **Etapa 4 implementación:** Para la fase de implementación hemos de seleccionar las herramientas adecuadas, un entorno de desarrollo que facilite nuestro trabajo y un lenguaje de programación apropiado para el tipo de sistema que vayamos a construir. La elección de estas herramientas dependerá en gran parte de las decisiones de diseño que hayamos tomado hasta el momento y del entorno en el que nuestro sistema deberá funcionar.

Además de las tareas de programación asociadas a los distintos componentes de nuestro sistema en la fase de implementación también hemos de encargarnos de la adquisición de todos los recursos necesarios para que el sistema funcione, por ejemplo, licencias de uso del sistema gestor de base de datos que vayamos a utilizar, etc. Usualmente también desarrollaremos algunos casos de prueba que nos permitan ir comprobando el funcionamiento de nuestro sistema conforme vayamos construyéndolo.

- **Etapa 5 testing y mantención:** Esta etapa se descompone en dos subetapas que se detallan a continuación:

- **Pruebas:** La etapa de prueba tiene como objetivo detectar los errores que se hayan podido cometer en las etapas anteriores del proyecto. Esto se hace antes de que el usuario final del sistema lo tenga que usar.

Una prueba es un éxito cuando detecta un error, y no al revés, como nos gustaría pensar; la búsqueda de errores se realiza en la etapa de prueba y pueda adoptar distintas formas en función del contexto y las fases del proyecto en la que nos encontramos. Algunas pruebas que se deben realizar son las siguientes:

- **Pruebas de Unidad**
- **Pruebas de Integración**
- **Pruebas Alfa**
- **Pruebas Beta**
- **Pruebas de Aceptación**

Una vez concluidas las etapas del desarrollo de un sistema de información análisis, diseño, implementación y prueba, llega el instante de poner el sistema en funcionamiento, y resulta esencial que tengamos en cuenta las dependencias que puedan existir entre los distintos componentes del sistema y sus versiones.

- **Mantenimiento:** La etapa de mantenimiento consume típicamente del 40% al 80% de los recursos de una empresa de desarrollo de software. Es probablemente la etapa más importante del ciclo de vida del software dada su naturaleza, que ni se rompe ni se desgasta con el uso. Su mantenimiento incluye tres facetas diferentes:

- **Mantenimiento correctivo:** Eliminación de los defectos que se detectan durante su vida útil.
- **Mantenimiento adaptativo:** Cuando el sistema ha de funcionar sobre una nueva versión del sistema operativo o un entorno hardware diferente, se debe adaptar a estas nuevas necesidades.
- **Mantenimiento perfectivo:** Cuando se proponen características nuevas que supondrían una mejora del sistema ya existente, se deben añadir nuevas funcionalidades

Si examinamos las tareas que se llevan a cabo durante la etapa de mantenimiento, nos encontramos aquí en el mantenimiento se repiten todas las etapas que ya hemos visto del ciclo de vida de un sistema de información.

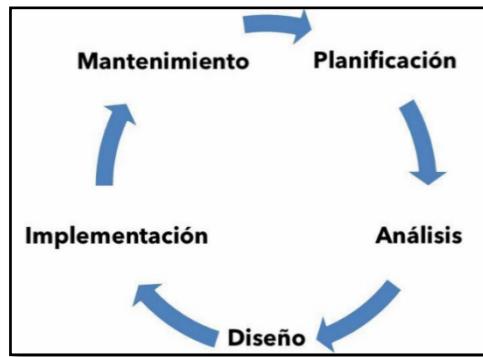


Figura N° 6: Ciclo de Vida de Sistemas
 (Tecnologías Información, 2018)

En el área de desarrollo de sistemas, hablar del uso de una metodología es lo mismo que hablar del ciclo de vida del desarrollo de sistemas. Ya que las fases que contiene el ciclo de vida se llevan a cabo en gran parte de las metodologías.

El ciclo de desarrollo de sistemas contempla siete fases partiendo por la fase 1 de identificar problemas, oportunidades y objetivos. Fase 2 incluye la determinación de los requerimientos humanos de información. Fase 3 se enfoca en el análisis de las necesidades del sistema. Fase 4 en el diseño del sistema. Fase 5 en el desarrollo y documentación del software. Fases 6 Prueba y mantenimiento del sistema. Fase 7 implementación y evaluación del sistema. Aunque cada fase se presenta de manera discreta, en realidad nunca se puede llevar a cabo como un paso separado, sino que varias actividades pueden ocurrir al mismo tiempo, e incluso se pueden repetir.

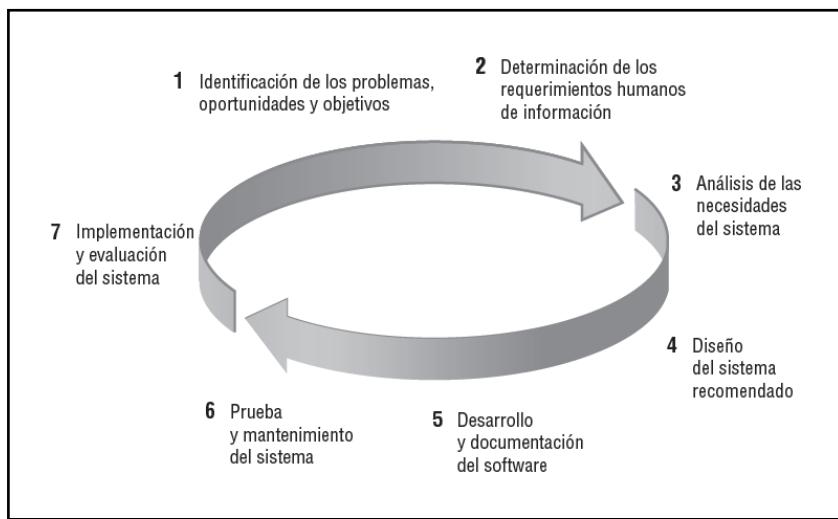


Figura N° 7: Las 7 Fases del ciclo de desarrollo de sistemas (SDLC)
 (Kendall & Kendall, 2011)

2.3. Problemáticas asociadas al desarrollo de sistemas

El software hoy en día es el producto más importante que podemos desarrollar ya que maneja grandes cantidades de información como por ejemplo datos personales transacciones financieras, etc. además permite administrar los negocios de manera eficiente. Cuando un sistema tiene éxito la sofisticación y complejidad producen resultados deslumbrantes, pero también plantean problemas enormes para que ellos que deben construir sistemas complejos.

La problemática surge cuando comienzan a aparecer algunas las siguientes preguntas

- ¿Por qué se requiere tanto tiempo para terminar el software?
- ¿Por qué son tan altos los costos de desarrollo?
- ¿Por qué no podemos detectar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué dedicamos tanto tiempo y esfuerzo a mantener los programas existentes?
- ¿Por qué seguimos con dificultades para medir el avance mientras se desarrolla y mantiene el software?

Los proyectos de desarrollo de software tienen siempre riesgo de que no resulten como se esperaba, sin embargo, la pregunta es ¿Por qué tantos proyectos fallan?

Aunque muchos podríamos pensar que la respuesta se encuentra en aspectos técnicos, la realidad es otra. En realidad, más del 50% fallan debido a aspectos relacionados con su gestión y formas de trabajo en el equipo e involucrados. Mientras que un bajo porcentaje es debido a problemas técnicos.

2.3.1. Las razones principales por la que los proyectos fallan

- **Inadecuada gestión del proyecto de desarrollo de software:** El no tener definidos y controlados elementos cruciales del proyecto como: Requerimientos, planes, seguimiento, control de cambios, control de calidad, recursos requeridos, entre otros. Esto dará como resultado una mala gestión de nuestro proyecto de desarrollo.
Derivado de esto veremos cuestiones como no lograr las fechas compromiso, tener un continuo cambio de alcance del proyecto, retrabajo y numerosos errores en las liberaciones que realicemos. Para evitar esto es recomendable adoptar una metodología o modelo para nuestro proceso.
- **Falta de comunicación del equipo de desarrollo de software y los demás:** El tener una pobre comunicación, así como carecen de los mecanismos para propiciarla dará como resultado huecos y omisiones en el proyecto.

En muchos de los casos, esto dará como consecuencia, la falta de definiciones que comúnmente deberán hacer los involucrados no técnicos y usuarios de negocio al equipo de desarrollo de software.

Será, por lo tanto, vital incluir tanto al equipo técnico como no técnico, a miembros con habilidades sólidas de comunicación.

De ellos se puede definir:

- **Correo electrónico**
- **Software de mensajería como Skype o Slack**
- **Reuniones de trabajo o seguimiento al proyecto**
- **Conferencias remotas**
- **Registros o bitácoras de actividades**
- **Entre otras**

Algunos de los documentos que son recomendables realizar para evidenciar y obtener el visto bueno del entendimiento son:

- **Documento de visión del negocio**
- **Historias de usuario y casos de uso**
- **Prototipos**
- **Documento de definición reglas de negocio**
- **Modelo de procesos**
- **Secuencias o actividades**
- **Diagramas de contexto**

- **Falta de compromiso del equipo de proyecto:** En la mayoría de los casos ocurre porque se cree que el área de sistemas o de tecnología, o el equipo de desarrollo de software es el responsable total y completo del éxito o fracaso del proyecto. Esto es un error que cometen los niveles directivos o gerenciales al dejar toda la responsabilidad solamente a dicha área o conjunto de personas.

El equipo de proyecto debe estar conformado tanto por el personal que construirá el producto en cuestión, así como por los representantes de usuarios y personas que darán visto bueno y validación al sistema. Para resolverlo se requiere que antes del inicio del proyecto, se identifiquen quiénes son los diferentes responsables. Una forma de hacerlo es a través de definir una matriz de responsabilidades RACI por sus siglas en inglés de Responsible, Accountable, Consulted e Informed.

- **Resultados pobremente definidos o sin definir:** Aunque esto parece increíble, muchos de los proyectos, no tienen claramente sus resultados definidos. Así como es preciso definir con total precisión los requerimientos técnicos, diseños, guías y manuales, es primero indispensable ver qué resultados queremos conseguir. Estos deben ser precisos y carecer de ambigüedad, dado que serán en realidad lo que dará guía y dirección a todo lo que se especifique en nuestro desarrollo de software.

Una de las mejores formas para documentar y dejar claro lo que queremos conseguir como resultados de negocio, es a través de un documento de visión.

- **No implementar pruebas o revisiones:** El no planear y ejecutar pruebas consistentes de las funcionalidades desarrolladas en el producto de software es un gran problema. Dado que los sistemas son diseñados para ejecutarse por personas y no por máquinas, el no validar su funcionamiento y entendimiento, dará posibilidad a que el software no se desarrolle como es requerido.

La gran mayoría de los programadores e ingenieros creen saber lo que el usuario quiere, pero esto puede diferir mucho de lo que en realidad son sus necesidades y problemas. Por lo tanto, es indispensable conducir en todo momento pruebas y verificaciones tanto al software, como a la especificación de este, respectivamente. Las pruebas no se deben detener nunca y es obligación del equipo de proyecto realizarlas en todo momento, así como asegurarse de tener el visto bueno del usuario.

El resultado debe ser algo parecido a esto...

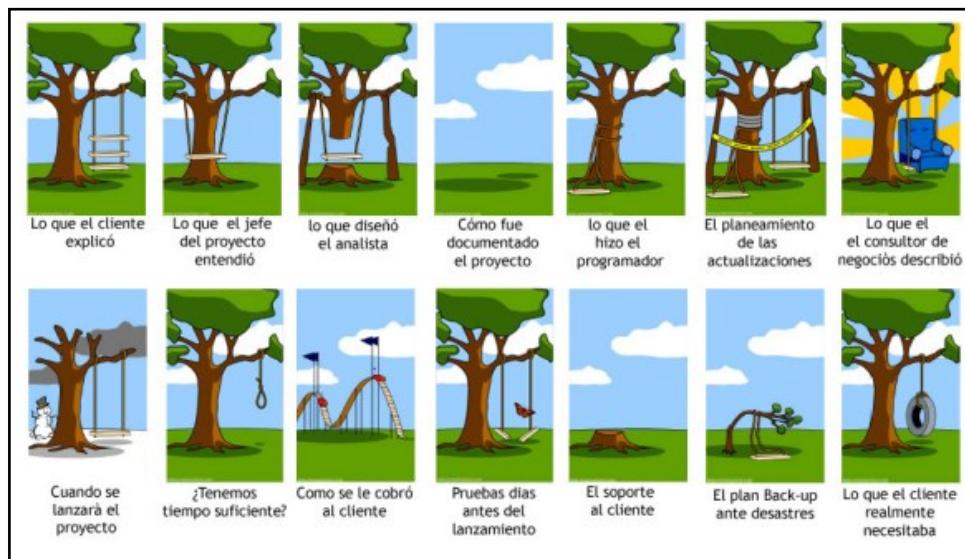


Figura N° 8: Problemáticas del desarrollo de software
 (cemeblog, s.f.)

2.4. Objetivos asociados al ciclo de vida

Cada etapa del ciclo de vida debe terminar cuando se alcanza un conjunto bien definido de objetivos en términos de producción, avance y entregas acordados previamente con los involucrados del proyecto según lo que se haya indicado en la planificación inicial.

A esto llamamos objetivos del ciclo de vida que marcan el éxito según la fase del proyecto en la que nos encontramos, los objetivos van a ser diferentes, por ejemplo.

- **Objetivos del ciclo de vida de la fase de concepción:** Documento visión, modelo de casos de uso, documento de requisitos suplementarios.

- **Objetivos del ciclo de vida de la fase de elaboración:** Documentos de arquitectura del sistema, prototipo de interfaz de usuario, prototipo técnico de la arquitectura del software.
- **Objetivos del ciclo de vida de la fase de construcción:** Versión beta del software en ejecución en ambiente de pruebas, documentación de usuario, soporte de entrenamiento de usuario, etc.

El objetivo del ciclo debe ser concreto, verificable, planificado de antemano, bien conocido y aceptado por todos los stakeholders del proyecto.

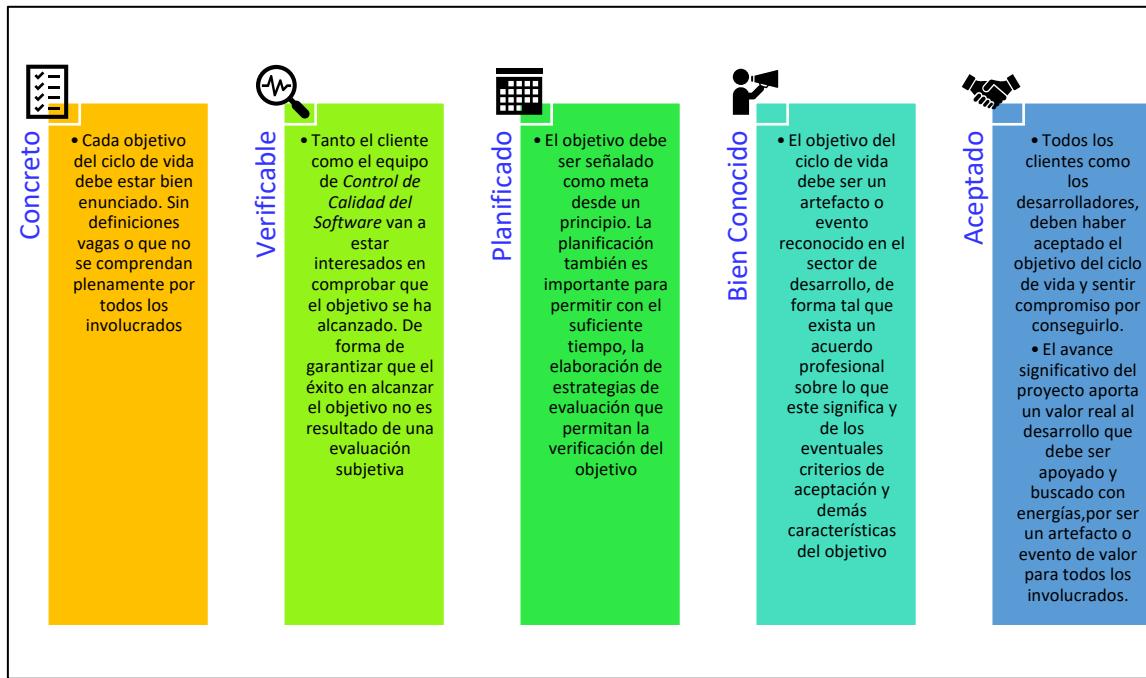


Figura N° 9: Criterios de los Objetivos del Ciclo de Vida
 (Tecnología y Synergix, s.f.)

2.4. Metodologías ágiles: Scrum y XP

La agilidad se ha convertido en la palabra mágica de hoy para describir un proceso del software moderno. Un equipo ágil es capaz de responder de manera apropiada a los cambios. Hay cambios en el software que se construye, en los miembros del equipo, debidos a las nuevas tecnologías y que tienen un efecto en el producto que se elabora o en el proyecto que lo crea. Un equipo ágil reconoce que el software es desarrollado por individuos que trabajan en equipo, y que su capacidad, su habilidad para colaborar y la comunicación entre los miembros del equipo es el fundamento para el éxito del proyecto.

La agilidad puede aplicarse a cualquier proceso del software. Sin embargo, para lograrlo es esencial que este se diseñe en forma que permita al equipo del proyecto adaptar las tareas y hacerlas directas, ejecutar la planeación de manera que entienda la fluidez de un enfoque ágil del desarrollo, eliminar todos los productos del trabajo excepto los más esenciales y mantenerlos

simples, y poner el énfasis en una estrategia de entrega incremental que haga trabajar al software tan rápido como sea posible para el cliente, según el tipo de producto y el ambiente de operación. Dentro del contexto de la metodología ágil, las más conocidas y utilizadas son Scrum y XP.

2.4.1. Scrum

Scrum es una metodología ágil desarrollada a principios de la década de 1990 la cual se enfoca en un proceso de análisis e incorpora las siguientes actividades estructurales: Requerimientos, análisis, diseño, evolución y entrega. Dentro de cada actividad estructural, las tareas del trabajo ocurren en patrones del proceso llamado sprint. El trabajo realizado dentro de un sprint se adapta al problema en cuestión y se define y con frecuencia se modifica en tiempo real por parte del equipo scrum. En la Figura N° 10 se muestra el flujo general del proceso scrum.

Scrum acentúa el uso de un conjunto de patrones de proceso de software que pone énfasis en las prioridades del proyecto, las unidades de trabajo agrupadas, la comunicación y la retroalimentación frecuente con el cliente. Scrum ha demostrado ser eficaz para proyectos cortos con tiempos de entrega muy apretados, con requerimientos cambiantes y para negocios críticos.

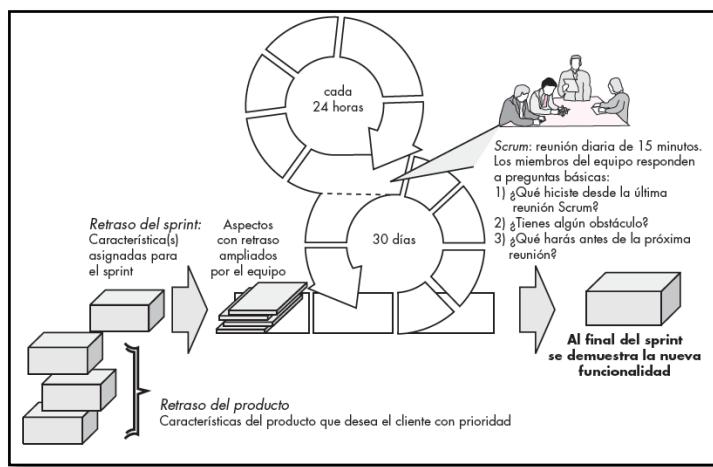


Figura N° 10: Flujo de Proceso Scrum
 (Pressman, 2010)

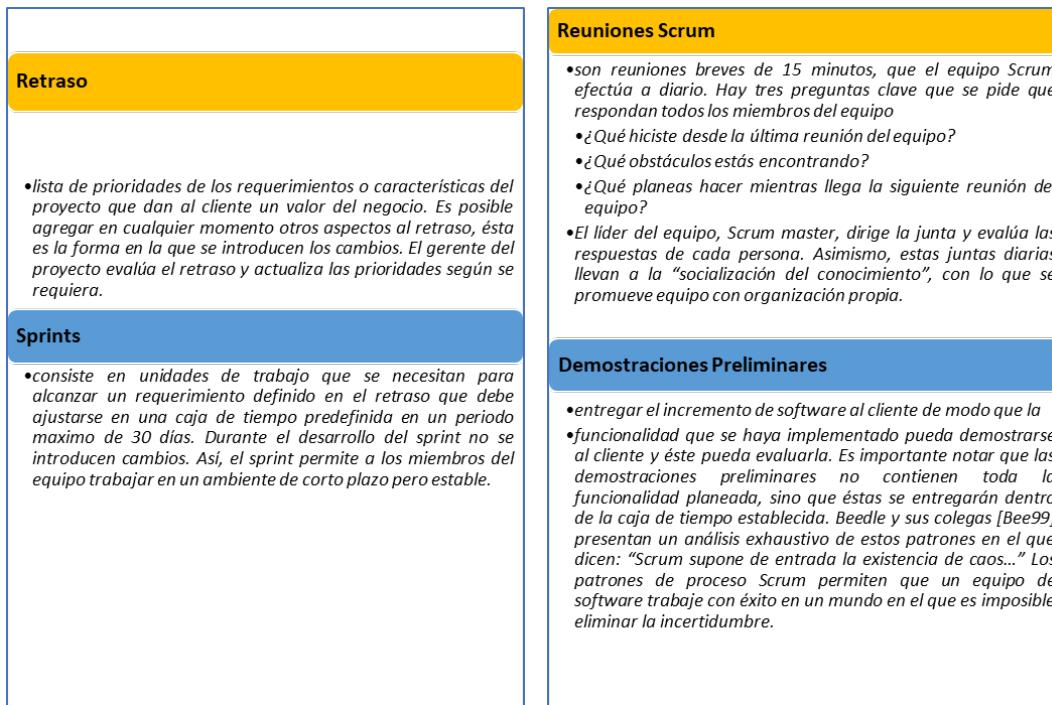


Figura N° 11: Elementos de Scrum
 (Propia, Elaboración, s.f.)

2.4.2. XP

La programación extrema (XP) es quizás el método ágil mejor conocido y más ampliamente usado. El nombre lo acuñó Beck (2000) debido a que el enfoque se desarrolló llevando a niveles “extremos” las prácticas reconocidas, como el desarrollo iterativo. Por ejemplo, en la XP, muchas versiones actuales de un sistema pueden desarrollarse mediante diferentes programadores, integrarse y ponerse a prueba en un solo día.

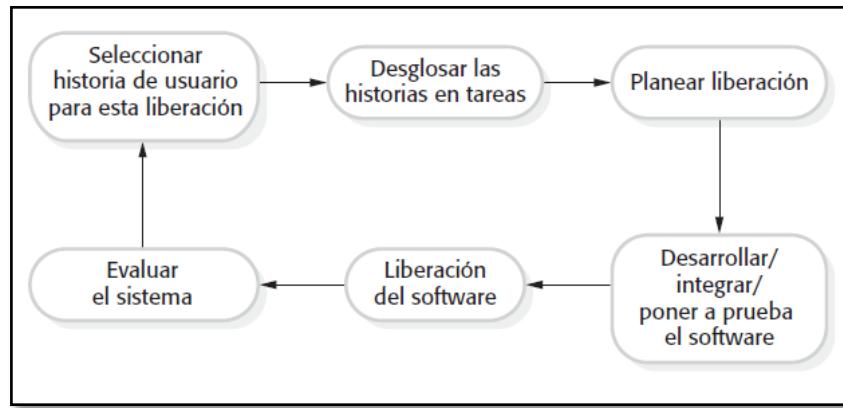


Figura N° 12: El ciclo de liberación de la programación extrema
 (Pressman, 2010)

XP se propuso en una época más reciente tiene como objetivo el proceso ágil para ser usado específicamente en organizaciones grandes. XP, o también conocida como programación extrema, define un conjunto de cinco valores que establecen el fundamento para todo trabajo realizado como parte de XP: Comunicación, simplicidad, retroalimentación, valentía y respeto. Cada uno de estos valores se usa como un motor para actividades, acciones y tareas específicas de XP.



Figura N° 13: Los cinco valores de XP

(Pressman, 2010)

2.5. Metodologías de desarrollo: Cascada, Incremental y de Integración

Los modelos de proceso prescriptivo fueron propuestos originalmente para poner orden en el caos del desarrollo de software. La historia indica que estos modelos tradicionales han dado cierta estructura útil al trabajo de ingeniería de software y que constituyen un mapa razonablemente eficaz para los equipos de software. Sin embargo, el trabajo de ingeniería de software y el producto que genera siguen “al borde del caos”.

Si los modelos de proceso prescriptivo buscan generar estructura y orden, ¿son inapropiados para el mundo del software, que se basa en el cambio? Pero si rechazamos los modelos de proceso tradicional (y el orden que implican) y los reemplazamos con algo menos estructurado, ¿hacemos imposible la coordinación y coherencia en el trabajo de software? No hay respuestas fáciles para estas preguntas, pero existen alternativas disponibles para los ingenieros de software. Se llaman “prescriptivos” porque prescriben un conjunto de elementos del proceso: Actividades estructurales, acciones de ingeniería de software, tareas, productos del trabajo, aseguramiento de la calidad y mecanismos de control del cambio para cada proyecto. Cada modelo del proceso también prescribe un flujo del proceso llamado flujo de trabajo, es decir, la manera en la que los elementos del proceso se relacionan entre sí.

2.5.1. Cascada¹⁰

El modelo de ciclo de vida clásico, también denominado "modelo en cascada", se basa en intentar hacer las cosas bien desde el principio, de una vez y para siempre. Se pasa, en orden, de una etapa a la siguiente solo tras finalizar con éxito las tareas de verificación y validación propias de la etapa (Figura N° 14). Si resulta necesario, únicamente se da marcha atrás hasta la fase inmediatamente anterior.

Este modelo tradicional de ciclo de vida exige una aproximación secuencial al proceso de desarrollo del software. Por desgracia, esta aproximación presenta una serie de graves inconvenientes, entre los que cabe destacar:

- *Los proyectos reales raramente siguen el flujo secuencial de actividades que propone este modelo.*
- *Normalmente, es difícil para el cliente establecer explícitamente todos los requisitos al comienzo del proyecto (entre otras cosas, porque hasta que no vea evolucionar el proyecto no tendrá una idea clara de qué es lo que realmente quiere).*

No habrá disponible una versión operativa del sistema hasta llegar a las etapas finales del proyecto, por lo que la rectificación de cualquier decisión tomada erróneamente en las etapas iniciales del proyecto supondrá un costo adicional significativo, tanto económico como temporal y eso sin tener en cuenta la mala impresión causada por un retraso en la fecha de entrega.

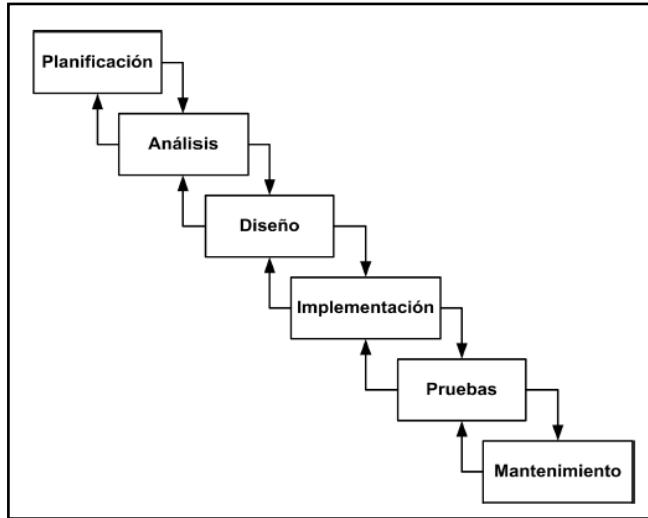


Figura N° 14: El ciclo de vida clásico: Modelo "en cascada"
 (Pressman Roger S., 2005)

¹⁰ Ingeniería de Software “Un Enfoque práctico”, Roger Pressman, Capítulo 2 “Modelos del Proceso”, 2.3.1 “Modelo de la Cascada”

2.5.2. Incremental¹¹

Hay situaciones en las que los requerimientos iniciales del software están razonablemente bien definidos, pero el alcance general del esfuerzo de desarrollo imposibilita un proceso lineal. Además, tal vez haya una necesidad urgente de dar rápidamente cierta funcionalidad limitada de software a los usuarios y aumentarla en las entregas posteriores. En tales casos, se elige un modelo de proceso diseñado para producir el software en incrementos.

El modelo incremental combina elementos de los flujos de proceso lineal y paralelo. En relación con la Figura N° 15, el modelo incremental aplica secuencias lineales en forma escalonada a medida que avanza el calendario de actividades. Cada secuencia lineal produce “incrementos” de software susceptibles de entregarse de manera parecida a los incrementos producidos en un flujo de proceso evolutivo.

El modelo de proceso incremental se centra en que en cada incremento se entrega un producto que ya opera. Los primeros incrementos son versiones desnudas del producto final, pero proporcionan capacidad que sirve al usuario y también le dan una plataforma de evaluación. El desarrollo incremental es útil en particular cuando no se dispone de personal para la implementación completa del proyecto en el plazo establecido por el negocio. Los primeros incrementos se desarrollan con pocos trabajadores. Si el producto básico es bien recibido, entonces se agrega más personal si es necesario para que trabaje en el siguiente incremento. Además, los incrementos se planean para administrar riesgos técnicos. Por ejemplo, un sistema grande tal vez requiera que se disponga de hardware nuevo que se encuentre en desarrollo y cuya fecha de entrega sea incierta. En este caso, tal vez sea posible planear los primeros incrementos de forma que eviten el uso de dicho hardware, y así proporcionar una funcionalidad parcial a los usuarios finales sin un retraso importante.

¹¹ Ingeniería de Software “Un Enfoque práctico”, Roger Pressman, Capítulo 2 “Modelos del Proceso”, 2.3.2 Modelos de Proceso Incremental.

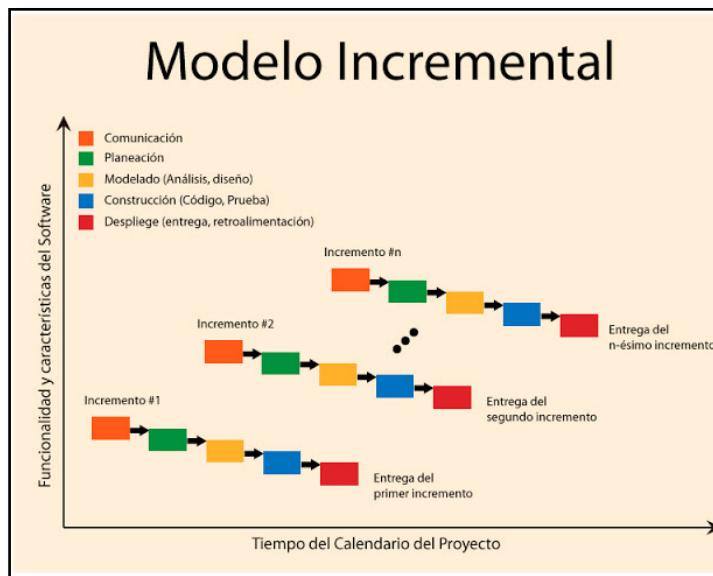


Figura N° 15: *Modelo Incremental*
 (Pressman, 2010)

2.6. Iteración¹²

Es un modelo derivado del ciclo de vida en cascada. Busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malentendidos durante la etapa de recogida de requisitos. Consiste en la iteración de varios ciclos de vida en cascada. Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto. El cliente es quien después de cada iteración evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga las necesidades del cliente.

Este modelo se suele utilizar en proyectos en los que los requisitos no están claros por parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente.

¹² Pressman, R. S. (1992). Ingeniería del software. McGraw-Hill Interamericana.

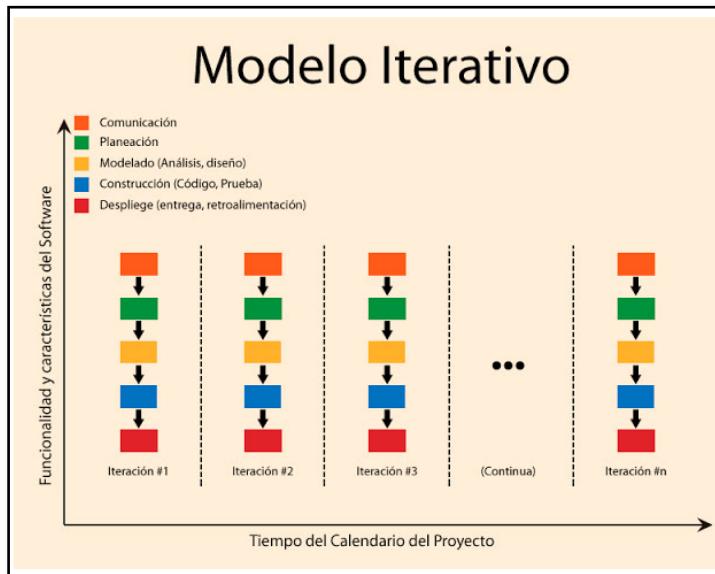


Figura N° 16: Modelo Iterativo
(Pressman Roger S., 2005)

Actividad del Aprendizaje Esperado N° 2

Actividad N° 2

Describa brevemente qué significan los siguientes términos:

- a) **Metodología de desarrollo de software**

- b) **Modelo de Ciclo de Vida para el desarrollo de software**

- c) **Etapa dentro de un ciclo de vida**

d) **Rol** que puede cumplir una persona en el desarrollo de software

e) **Modelo/diagrama** de las características de un sistema de software y sus partes componentes

Aprendizaje Esperado 3

Analizan sistema mediante modelos entidad-relación, considerando organización de entidades, sistemas ERP y objetivos de herramientas de software.

3.1. Estructura y concepto de las organizaciones

La estructura organizacional es el sistema jerárquico escogido para organizar el personal y los medios de una organización. Gracias a ella, se abordan las formas de organización interna y administrativa. El reparto del trabajo en áreas o departamentos se ramifica en un árbol.

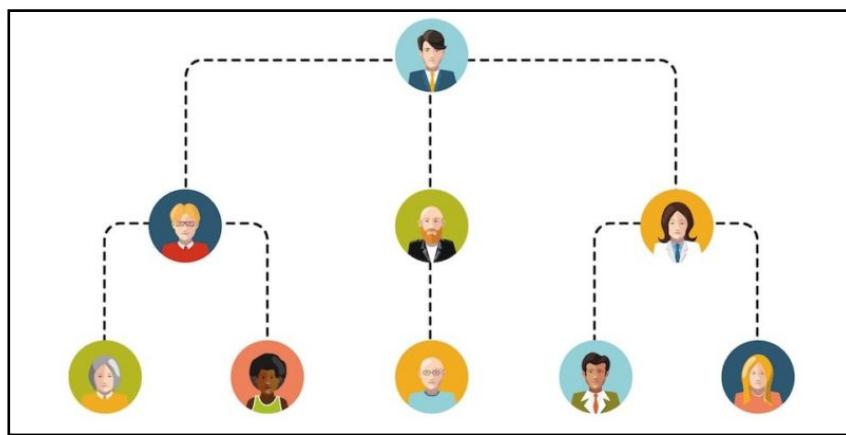


Figura N° 17: Estructura Organizacional
(Bizneo Blog, s.f.)

Se podría decir que la estructura de la organización es el modo de planificar su trabajo y repartir formalmente sus responsabilidades. Sin embargo, es muy importante que el acto de organizar dé como resultado una estructura de la organización, estructura que pueda considerarse como el marco de trabajo que retiene unidas las diversas funciones de acuerdo con un esquema, que sugiere orden y relaciones armoniosas.

Estas estructuras pueden ser de dos tipos:

- **Centralizado.** Concentra las decisiones en los altos cargos de la jerarquía.
- **Descentralizado.** Permite a cada departamento importantes márgenes de autonomía.

Existen cuatro elementos básicos en la estructuración de las organizaciones:

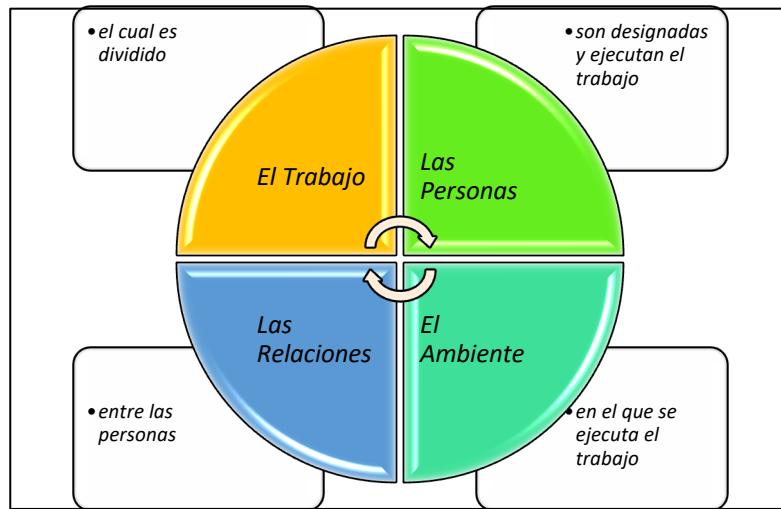


Figura N° 18: Elementos básicos de las Organizaciones
 (Propia, Elaboración, s.f.)

3.1.1. Elementos clave de la estructura organizacional¹³

Toda estructura organizacional cuenta con algunos componentes fundamentales que las distinguen de otras.

- **Cadena de Mando:** Es la base de todo modelo organizacional. Se trata de una línea de autoridad que fluye desde la más alta dirección hasta los puestos más bajos. Esta cadena define a quién hay que dirigirse según el tema que se trate. Para evitar las confusiones, es recomendable disponer de una plataforma que aclare la cadena de mando y mejore las comunicaciones internas.
- **Nivel de centralización:** Que la organización esté centralizada o descentralizada contribuirá directamente en la velocidad del proceso de toma de decisiones. También tendrá un impacto en la percepción más o menos democrática de la forma de actuar.
- **Margen de control:** Es un elemento muy influido por el tamaño de la compañía y el nivel de centralización. Cuantos más empleados controle un mando directivo, mayor será su margen de control.
- **Grado de especialización:** Esta clave de la estructura organizacional aborda la división del trabajo. Dentro de una organización, las tareas se distribuyen en distintos niveles. Si los empleados cuentan con una especialización alta, estos serán expertos en su campo y serán más productivos. Sin embargo, al mismo tiempo, cuando el grado de especialización es menor, la relación es más flexible y versátil. Ambas opciones tienen sus ventajas y desventajas. Lo ideal es encontrar la más adecuada para la actividad de la empresa, con cierto equilibrio entre ellas.

¹³ 6 elementos clave de la estructura organizacional; Bizneo HR Software <https://www.bizneo.com/blog/estructura-organizacional/>

- **Formalidad estructural:** Otro punto sobre el que hay que mantener cierta armonía es el grado de formalidad de la estructura organizacional. Un compromiso entre rigidez y libertad que permita trabajar con procesos rápidos, pero sin eliminar las relaciones entre los trabajadores ni su creatividad.

- **Formación de departamentos:** Las diferentes actividades de una organización se dividen en departamentos, cada uno de ellos con sus propios proyectos. Cuando se construye el modelo organizacional, se estudia la necesidad de departamentalizar de forma rígida o flexible. Las estructuras más rígidas alcanzan mayor grado de especialización, mientras que las flexibles incentivan la colaboración entre departamentos.

3.2. Sistemas empresariales ERP

El término ERP se refiere a Enterprise Resource Planning, que significa “sistema de planificación de recursos empresariales”. Estos programas se hacen cargo de distintas operaciones internas de una empresa, desde producción a distribución o incluso recursos humanos.

Los sistemas ERP suponen una gran inversión para las empresas. Según una encuesta de Panorama Consulting de 2013, un 40% de las empresas que adquieren un ERP notan un aumento la productividad.

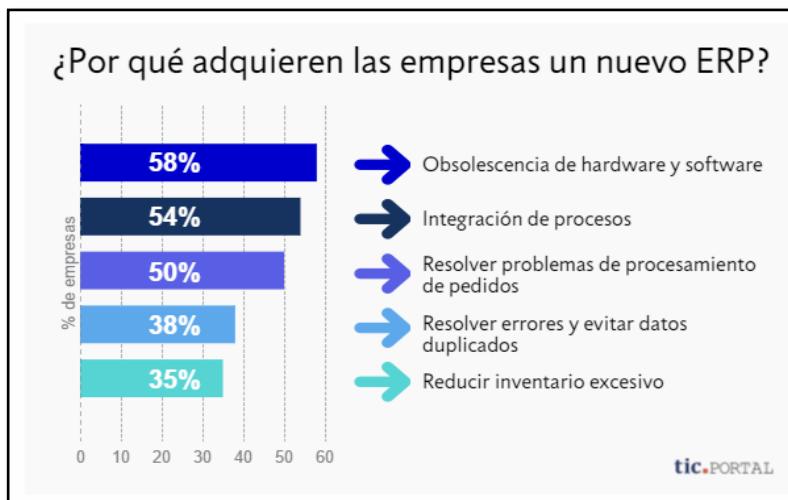


Figura N° 19: Gráfico de datos de «Microsoft Dynamics Enterprise Applications for SMB». (EKCIT, s.f.)

3.2.1. Características del ERP¹⁴

Las dos características principales que diferencian a los ERP de otros programas de gestión empresarial es que son modulares y configurables:

1. **Modulares:** Los ERP cuentan con diferentes programas o módulos que gestionan los diferentes departamentos de la empresa, tales como ventas, marketing, almacenes o recursos humanos.

¹⁴ Iniciativa de EKCIT; Ticportal; Guía ERP 2021, Que es un sistema ERP <https://www.ticportal.es/temas/enterprise-resource-planning/que-es-sistema-erp>

Todos estos módulos comparten información en torno a una base de datos común que vertebría el funcionamiento del ERP.

2. **Configurables:** Los ERP deben poder modificarse para adaptarse a las necesidades específicas de cada empresa, por ejemplo, a la hora de gestionar el inventario o los puntos de venta. Por eso es necesario que el ERP se pueda configurar para adaptarse a diferentes organizaciones y procesos, teniendo en cuenta además que las necesidades de una misma empresa varían a través del tiempo.

Entre las soluciones informáticas más comunes para las empresas, podemos encontrar múltiples ejemplos de ERP. Un ERP es un software de gestión empresarial que permite planificar y controlar los procesos y recursos de negocio de una empresa.



Figura N° 20: Tipos de ERP
(Aner Technology , s.f.)

3.2.2. Diferentes programas ERP

- [Microsoft Dynamics 365](#)
- [Microsoft Dynamics NAV 2016](#)
- [Microsoft Dynamics NAV 2015](#)
- [Microsoft Dynamics AX \(2016\)](#)
- [Microsoft Dynamics AX 2012](#)
- [SAP Business One](#)
- [SAP Business All-in-One](#)
- [SAP Business ByDesign](#)
- [NetSuite OneWorld](#)
- [Oracle J.D. Edwards EnterpriseOne](#)
- [Oracle EBS \(E-Business Suite\)](#)

3.3. Niveles de organización¹⁵

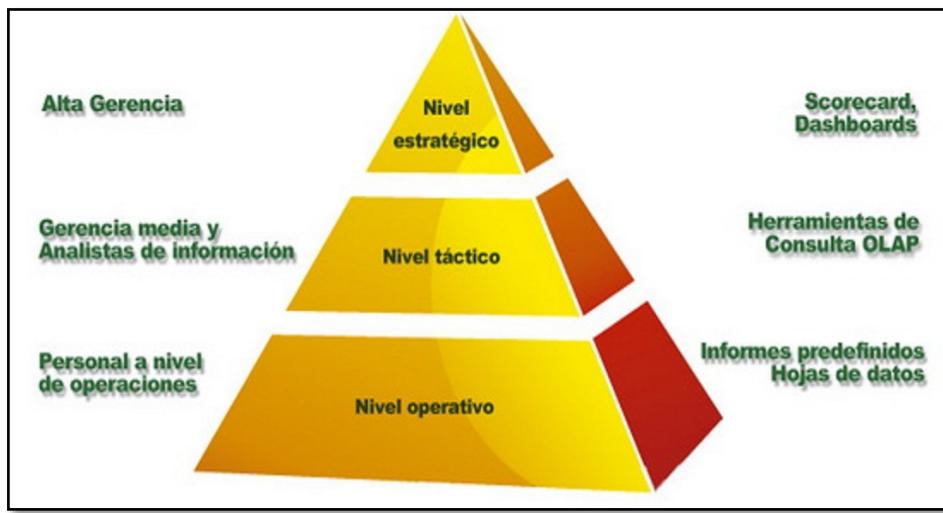


Figura N° 21: Niveles de Organización Empresarial o Administrativa
 (apd, s.f.)

Los niveles de gestión empresarial son una garantía del éxito y de la eficacia de estas organizaciones. En total existen tres niveles (Figura N° 21) que ayudan a planificar las líneas maestras que moverán la actividad empresarial y qué se hará para mantener el rumbo.

Cada nivel cumple con una serie de tareas que se complementan para garantizar el éxito de toda la organización. Si uno de ellos desapareciera, toda la estructura se resentiría, ya que no habría un plan ni objetivos que seguir.

1. Nivel estratégico:¹⁶ Se trata de la visión que mueve las acciones de la empresa. Establece los objetivos a cumplir y las líneas maestras para alcanzarlos. La dirección juega un rol principal a la hora de definir la estrategia, por lo que debe actuar con suma precisión para que toda la organización comprenda su visión.

2. Nivel táctico: Solo los departamentos se encargan de desarrollar este nivel. Se crean las acciones a realizar para hacer realidad la estrategia de la empresa. Es un tipo de planificación específica y que atiende en profundidad a los detalles.

3. Nivel operativo: En este último nivel aparecen los agentes encargados de ejecutar las acciones desarrolladas en el nivel táctico. Realizan acciones de corta duración y todos en la empresa tienen un rol que desempeñar en este nivel.

¹⁵ Apd por redacción APD 28/11/2019: <https://www.apd.es/niveles-gestion-empresarial/>

¹⁶ Lifeder.com Niveles Organizaciones o pirámide organización; <https://www.lifeder.com/niveles-organizacionales/>

3.4 Otros tipos de herramientas: CRM, BI, Punto de Ventas

Tipos de CRM

3.4.1. CRM operativo.

En el CRM operativo podemos diferenciar dos partes:

- Front Office que es la encargada de todos los elementos de la empresa que están en relación directa con el cliente: desde la gestión del marketing y las ventas hasta la atención al cliente.
- Back Office que es la parte encargada de los procesos organizativos que forman el entramado del negocio y que no están en contacto directo con el cliente. Está centrada en funciones de contabilidad y finanzas.

3.4.2. CRM analítico.

Conocido como Business Intelligence, consiste en recoger, transformar y poner a disposición de la empresa, información importante de los clientes. Este modelo se encarga del almacenamiento de datos, explotación e información de estos para poder ofrecerles un mejor servicio, medir y evaluar las campañas de marketing y conocer los canales de contacto preferidos por cada cliente son algunas de sus funciones. Son herramientas encargadas de conocer todo sobre las relaciones, tanto a nivel interno como a nivel externo.

3.4.3. CRM colaborativo

Se encarga de las interacciones entre la empresa y el cliente de esta manera, a la empresa le será más fácil establecer un vínculo con sus clientes, ofreciéndoles los servicios o productos que se adaptan a sus necesidades aprovechando la variedad de canales que ofrecen las nuevas tecnologías. Desde los puntos de venta físicos o a través de call center, hasta la interacción con los procesos a través de computadores o dispositivos móviles.

Lo que pretende el CRM colaborativo es establecer una comunicación multicanal tanto de forma interna como de forma externa y mejorar la relación con los clientes.

3.4.4. Gestión de relaciones con el cliente (CRM)

El término gestión de relaciones con el cliente o Client Relationship Management (CRM), es la metodología que se centra en almacenar, analizar y usar toda la información relevante de los clientes. Lo que se considera información relevante de los clientes, depende naturalmente del tipo de empresa y de clientes.



Figura N° 22: Ventajas de la gestión de relaciones con el cliente
 ((EKCIT, s.f.))

3.4.5. ¿Qué Clientes se deben incluir en el CRM?

El término “cliente” es la persona o empresa que compra un producto o contrata un servicio. Sin embargo, en esta definición se pueden entrever las divergencias de su significado. Además, otros tipos de relaciones también pueden gestionarse en un software CRM, como, por ejemplo, miembros o, incluso, partners comerciales.

Una organización afiliada adaptará su sistema más a la comunicación individual y la funcionalidad de marketing o ventas no es un requisito.

	Clientes comerciales				Miem-bros	Rela-cio-nales		
	Producción		Servicios					
	Esporá-dicos	Habi-tuales	Único	Recu-rrente				
Perfil individual del cliente		●		●	●	●		
Perfil general de los clientes	●		●					
Acuerdos realizados		●	●	●	●	●		
Fidelización clientes		●		●	●	●		
Fuerzas de ventas	●		●					
Información compartida (actividades, newsletters)					●	●		
Comunicación en ambos sentidos						●		

Figura N° 23: Prioridades del CRM dependiendo del tipo de relación
 ((EKCIT, s.f.))



Figura N° 24: Flujo de información entre departamentos con CRM
 ((EKCIT, s.f.))

3.4.6. BI (Business Intelligence) Inteligencia de negocios

El término Inteligencia de negocios es más reconocido por su nombre inglés, Business Intelligence (BI). Sirve a las empresas para analizar grandes cantidades de datos en “bruto”. Con esos datos se intenta obtener información valiosa para tomar decisiones comerciales estratégicas y operativas. Generalmente, los datos en bruto pueden ser datos de ventas, tendencias del mercado, resultados financieros, etc. provienen de los diferentes softwares de la compañía, tales como ERP, RRHH, gestión documental o software especializado.



Figura N° 25: BI a Nivel Organizacional
 (kaptadata, 2017)

Para llevar a cabo la inteligencia de negocios con éxito, se deben analizar suficientes datos. El paso a paso de BI sería así:

- I. **Se extraen los datos de los diferentes sistemas de software**
- II. **Se analizan los datos**
- III. **Se convierten los datos comerciales en bruto en información valiosa**
- IV. **Se actúa en consecuencia de las “ideas comerciales” recién adquiridas**

Los conceptos de almacenamiento de datos e integración de datos están relacionados con la inteligencia de negocios. Para sacar información útil a través de la inteligencia de negocios son de gran importancia tanto la calidad como la estructura de los datos. Con el análisis predictivo de BI, la compañía puede responder mejor a la situación futura de mercado.



Figura N° 26: Beneficios de Business Intelligence
(kaptadata, 2017)

Cada vez con mayor frecuencia se encuentra el software de BI integrado en los sistemas de ERP o de gestión de relación de clientes (CRM). Las principales empresas tienen ya instalados en su ERP un módulo de BI, como por ejemplo Microsoft (módulo de BI es PowerBI), Oracle (tiene 2, OBI e Hyperion) y SAP (se llama Business Objects).



Figura N° 27: Características de Business Intelligence
(kaptadata, 2017)

3.4.7. Componentes de BI



Figura N° 28: Componentes de BI
 (kaptadata, 2017)

3.4.9. Implantación de BI

1) **Dirigir y Planificar:** Esta fase involucra la redacción de requerimientos específicos y, a su vez, contesta preguntas que guían a otras que ayudarán a alcanzar objetivos.

2) **Recolección de Información:** Hay diversas fuentes de información dentro de una compañía, fuentes de recursos como son: Puntos de ventas, ERP, CRM, Aplicaciones de Servicios al cliente, etc. Este es un proceso continuo y es importante entender que los datos de esas fuentes son simplemente información y no Inteligencia. La información se convierte en Inteligente después de procesarla y de analizarla para determinar los datos necesarios para encontrar las respuestas a las preguntas.

3) **Procesamientos de Datos:** Esta fase es la integración de datos en crudo a un formato utilizable para el análisis, agregando datos a bases de datos nuevas o existentes o consolidando información. Generalmente es vista como Extracción, Transformación y Carga (ETL) que ocurre en los ambientes de BI.

4) **Análisis y Producción:** El grupo de análisis de negocios utiliza herramientas y técnicas para ordenar sobre los datos y crear inteligencia. El resultado final es la producción de respuestas “inteligentes”, en un contexto propio. En algunos casos es un proceso simple como la creación de un reporte. En otros casos, son la creación de indicadores.

5) **Difusión:** Esta fase de difusión es entregar productos inteligentes a los diversos clientes que lo requieren. Esto básicamente implica el uso de herramientas BI para la publicación de “tableros de indicadores”, reportes o la posibilidad de tener herramientas de fácil uso para que los mismos usuarios tengan la capacidad de revisar los datos de manera rápida y sencilla.



Figura N° 29: Etapas de Implementación de BI
(kaptadata, 2017)

3.4.10. Punto de Venta POS

Un Sistema Punto de Venta (POS por sus siglas en inglés) es comúnmente conocido como la caja registradora o terminal encargada de cobrar. Usualmente se asocia a una caja (el lugar donde se realiza el cobro) de cualquier establecimiento.

Sin embargo, en un término más estricto, el POS se compone de un software y hardware que te permite cobrar. Así que el conjunto de estos dos elementos es necesario para que se llame punto de venta y no simplemente se refiere a un aparato dedicado a cobrar, por ejemplo, una terminal punto de venta (TPV).

El primer POS que se tiene registrado es una caja registradora mecánica inventada en 1879. El propósito era mantener un mejor registro de lo que se vendió y el flujo de efectivo. El tiempo ha hecho que las máquinas sean digitales y así la mezcla de la máquina para cobrar y el software dedicado a cobrar conforman un POS.

Ahora, existen soluciones que integran todo en uno: Cobrar, realizar inventarios y ser un scanner de códigos de barras.



Figura N° 30: Sistemas POS Tradicionales
(SISTEMAPOS.NET, s.f.)

3.4.11. Sistema POS tradicional

Un sistema de POS tradicional es aquel que opera en una red única de computadores y que necesita instalarse de forma individual en cada equipo en el que se desee trabajar. Para usar estos sistemas tradicionales se requieren de una licencia y sus actualizaciones deben realizarse de forma manual.

Usualmente, los sistemas tradicionales suelen ser más costosos y necesitan de un servidor local para el almacenamiento de la información. Al igual que es necesaria la compra de equipos robustos para su funcionamiento.

3.4.12. Sistema POS en la nube

Los sistemas POS en la nube son la opción actual y más recomendada en POS, ya que funcionan a través de internet y permiten que los usuarios puedan acceder a él desde cualquier dispositivo, como tablet, computador o celular.

La ventaja que ofrece un sistema POS en la nube es que no requiere de ningún tipo de instalación y puede ser consultado desde cualquier lugar de manera rápida y con información actualizada.

Hay sistemas POS que funcionan incluso sin conexión a internet, debido a que son diseñados para solventar las fallas de conexión que se pudieran presentar con algunos usuarios. Con el POS sin conexión se pueden generar ventas y agregar pagos y en cuanto se restablezca la conexión a internet se sincronizarán automáticamente, generando incluso los reportes contables y administrativos.



Figura N° 31: Sistema POS en la Nube
(PROFIT SOS BLUE, s.f.)

3.4.13. Elementos que integran un Punto de Venta ideal

Para tener funcionando completamente **(01) Software Punto de Venta** necesitarás de ciertos complementos, estos son algunos:

1. **(02) Equipo de cómputo:** Aquí es donde se instala el Sistema de Punto de Venta que te ayuda para llevar el control de tu negocio. El equipo debe contar con conexiones USB para conectar los complementos como la impresora, el cajón de dinero y lector de código de barras.
2. **(03) Lector de código de barras:** Este elemento es opcional, sin embargo, facilitaría la operación diaria de tu negocio. Con un lector de código de barras podrás agilizar el tiempo de espera de

tus clientes para atender a más personas en menos tiempo pues solo necesitarás escanear el código de barras de cada producto y utilizarlo durante tus inventarios.

3. **(04) Impresora de tickets:** Es uno de los complementos indispensables para un Punto de Venta, con el podrás emitir los comprobantes de ventas, voucher de pago y/o reportes de corte de caja y más.
4. **(05) Cajón de dinero:** El cajón de dinero es una pieza esencial para tu Punto de Venta, pues te ayudará a organizar el dinero y por supuesto mantenerlo seguro, este espera la orden que le envíe la impresora para que pueda abrirse cuando es enviada a imprimir el ticket o nota de venta.
- **(06) Básculas electrónicas:** En caso de contar con una venta de productos a granel la báscula electrónica es un instrumento que necesitarás pues te permitirá obtener una indicación o lectura Digital con cifras exactas de los productos que estés vendiendo.

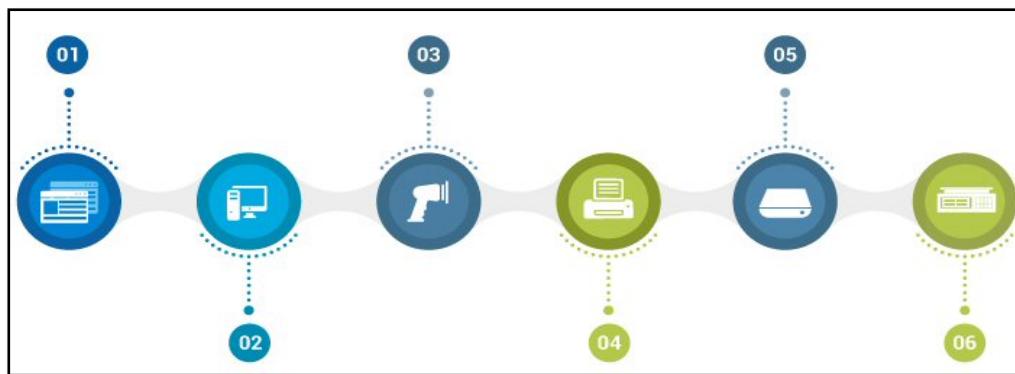


Figura N° 32: Elementos que integran un Punto de Venta
 (ManagementPro, s.f.)

¿Por qué las empresas escogen invertir en software para la gestión de clientes?

¿Para qué tipo de empresa es interesante?

¿Cómo se inicia una implementación de CRM?

3.5 Naturaleza y objetivos de sistemas SaaS

Un sistema SaaS o Software as a Service, es un modelo de distribución de software en el que tanto el software como los datos manejados son centralizados y alojados en un único servidor externo a la empresa. Esto implica que el software utilizado por la empresa no se encuentra en la misma, sino que un proveedor se ocupa del hosting de dicho software en la nube, así como del mantenimiento y el soporte.

La empresa contratante accede al software y todos sus datos a través de un navegador web desde cualquier computador. Eso quiere decir que toda la información, procesos, resultados, etc.

almacenados en este software son de fácil acceso desde cualquier lugar. Tanto el software como los datos están centralizados y hospedados en un único servidor.

La principal novedad es que el uso del software ya no se basa en la instalación de la aplicación en los computadores locales de la empresa. Cualquier trabajador puede acceder al programa desde cualquier punto del mundo sin necesidad de instalación previa, gran ventaja para empresas que operan a nivel global. Además, se reducen los costes de licencias, mantenimiento y soporte y se mejora la movilidad.

3.5.1 Los orígenes de SaaS

En la década de 1960, los computadores mainframe estaban conectados a terminales básicas que compartían el software de mainframe: Un sistema de prestación de software que se conoce como tiempo compartido. Como el costo de los computadores comenzó a bajar en la década de 1980, muchos negocios crearon su propia versión local de tiempo compartido, que se denominó red local (LAN). Sin embargo, el negocio -no el proveedor de tecnología- era responsable de suministrar y administrar el hardware y la red.

Con el advenimiento de internet en la década de 1990, los proveedores comenzaron a presentar el software y a hacerlo accesible a los clientes a través de internet. Sin embargo, este predecesor de SaaS, denominado modelo proveedor de servicios de aplicaciones (ASP), tenía graves limitaciones. Por ejemplo, cada cliente necesitaba su propia versión del software, lo que significaba que se tenía que instalar el software en los computadores de los usuarios. La configuración era costosa y llevaba mucho tiempo.

Las primeras soluciones SaaS surgieron a finales de los años 90, cuando se inventó originalmente el término SaaS. Este nuevo modelo proporcionaba un rendimiento mucho mayor que el modelo ASP. Una sola instancia de la aplicación podía servir a múltiples usuarios e incluso a los clientes, gracias a su arquitectura de tenencia múltiple. Ya no se necesitaba la instalación local del software. Y proporcionó un modo de recopilar, sumar y centralizar valiosos datos de aplicaciones.

3.5.2 ¿Cómo funciona el SaaS?

Por lo general, un proveedor de servicios de nube (como AWS, Azure o IBM Cloud) gestiona el entorno de nube en el cual se aloja el software. Las aplicaciones SaaS aprovechan la arquitectura multiempresa para utilizar los recursos agrupados. Además, el proveedor de SaaS se encarga de las actualizaciones del software, las correcciones de errores y demás tareas de mantenimiento general de las aplicaciones. Los usuarios interactúan con el software a través de un explorador web en sus computadores o dispositivos móviles. Además, pueden utilizar interfaces de programación de aplicaciones (API), como REST o SOAP, para conectarlo con otras funciones.

Debido a la naturaleza del SaaS, los proveedores pueden implementar funciones nuevas para los clientes con mucha más facilidad. La mayoría de las aplicaciones SaaS son productos plug and play

preconfigurados en los que el proveedor gestiona todos los elementos que respaldan la aplicación, entre los que se incluyen:

- Los elementos de hardware, como las redes, el almacenamiento y los servidores del centro de datos.
- Las plataformas, como la virtualización, el sistema operativo y el middleware
- Los requisitos de software, como los tiempos de ejecución, los datos y la aplicación misma.

3.5.3 Diferencias entre PaaS, IaaS y SaaS

Además del SaaS, las otras opciones como servicios principales son la Infraestructura como servicio (IaaS) y la Plataforma como servicio (PaaS).

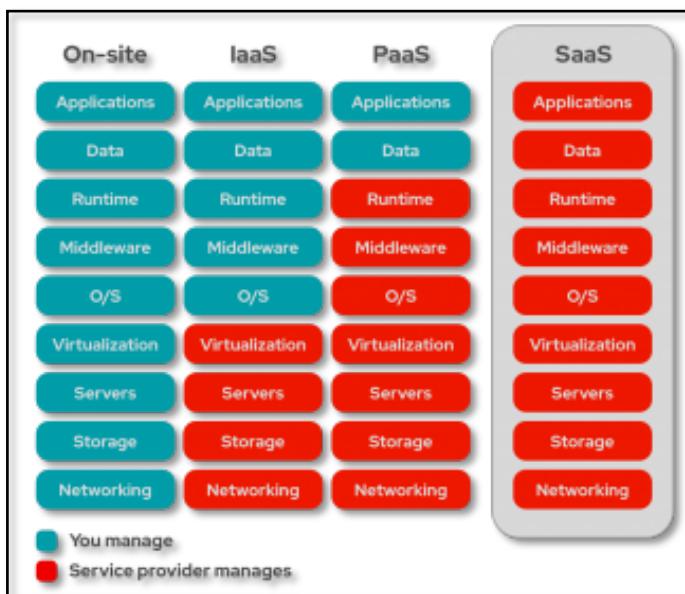


Figura N° 33 Servicios de SaaS
 (RedHat, s.f.)

3.5.3.1 Infraestructura como servicio (IaaS)

Esta es una oferta de los servicios en la nube en la que el proveedor facilita a los usuarios el acceso a herramientas como servidores, redes y almacenamiento de datos, así como copias de seguridad y máquinas virtuales. Esto ahorra a las empresas el coste de adquirir y sostener su propio hardware para estas operaciones.

3.5.3.3 Plataforma como servicio (PaaS)

Crea para los usuarios un entorno en la nube en el que pueden desarrollar aplicaciones. Además de las características que, ya mencionadas en el IaaS como el almacenamiento y capacidad de procesamiento de datos, ofrece herramientas prediseñadas para que los usuarios las personalicen y creen sus propias aplicaciones.

3.6 Diagramas de flujo

El diagrama de flujo, o también **diagrama de actividades**, es una manera de representar gráficamente un algoritmo o un proceso de alguna naturaleza, a través de una serie de pasos estructurados y vinculados que permiten su revisión como un todo.

Es una herramienta utilizada para representar la secuencia e interacción de las actividades del proceso a través de símbolos gráficos. Los símbolos proporcionan una mejor visualización del funcionamiento del proceso, ayudando en su entendimiento y haciendo la descripción del proceso más visual e intuitivo.

Hay cuatro tipos de diagrama de flujo en base al modo de su representación:

- **Horizontal:** Va de derecha a izquierda, según el orden de la lectura.
- **Vertical:** Va de arriba hacia abajo, como una lista ordenada.
- **Panorámico:** Permiten ver el proceso entero en una sola hoja, usando el modelo vertical y el horizontal.
- **Arquitectónico:** Representa un itinerario de trabajo o un área de trabajo.

Los diagramas de flujo son un mecanismo de control y descripción de procesos, que permiten una mayor organización, evaluación o replanteamiento de secuencias de actividades y procesos de distinta índole, dado que son versátiles y sencillos. Son empleados a menudo en disciplinas como la programación, la informática, la economía, las finanzas, los procesos industriales e incluso la psicología cognitiva.

3.6.1 Proceso de un Diagrama de Flujo

En este ámbito, hablamos de procesos para referirnos a una secuencia específica de actividades, es decir, a los pasos a dar dentro del diagrama de flujo. Por ejemplo, en informática, los procesos son secuencias iniciadas o bien por disparadores programados dentro del sistema, o por intervenciones del usuario del sistema. Cada uno posee una dirección, un propósito y una serie de pasos que abarca.

3.6.3 Simbología de un diagrama de flujo

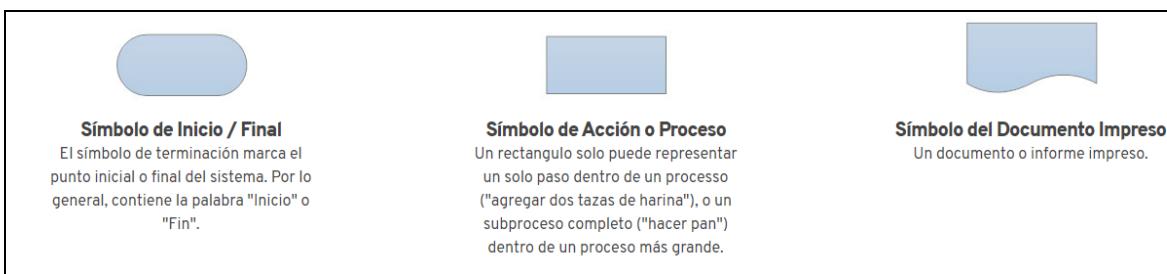
Los principales símbolos convencionales que se emplean en los diagramas de flujo son los siguientes:

Símbolo	Nombre	Función
	Inicio / Final	Representa el inicio y el final de un proceso
	Línea de Flujo	Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción.
	Entrada / Salida	Representa la lectura de datos en la entrada y la impresión de datos en la salida
	Proceso	Representa cualquier tipo de operación
	Decisión	Nos permite analizar una situación, con base en los valores verdadero y falso

Figura N° 34: Símbolos del Diagrama de Flujo
 (SmartDraw, s.f.)

Con los años, la tecnología ha evolucionado, y con ella también la diagramación. Algunos símbolos de los diagramas de flujo que se utilizaron en el pasado para representar tarjetas perforadas de computadora, o cinta perforada, han pasado a la historia.

Aquí tienen una lista más completa de los símbolos de diagramas de flujo. Cabe destacar que según la complejidad del proceso se pueden ir utilizando.



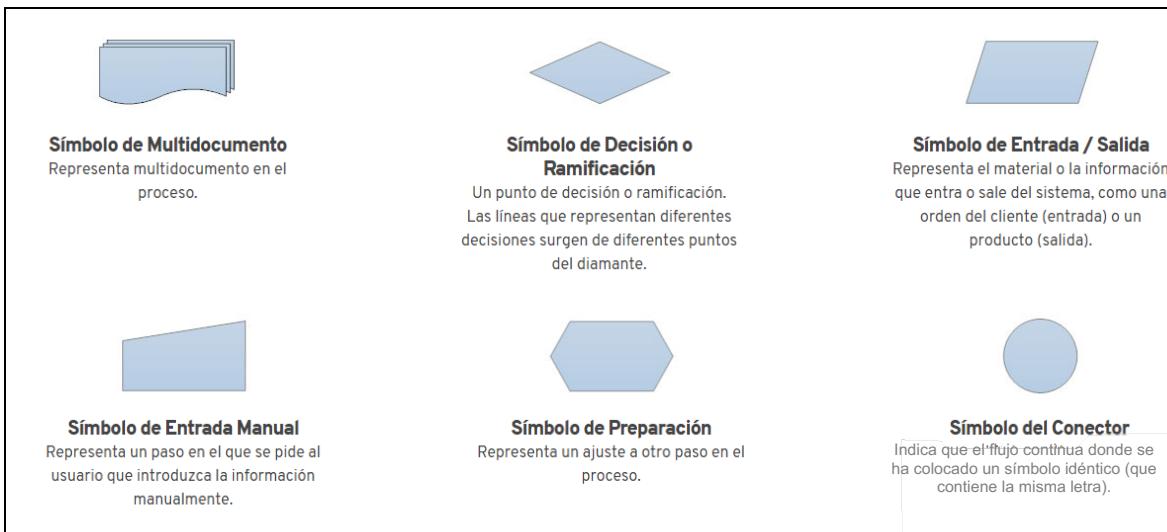


Figura N° 35: Símbolos más utilizados en DF
(SmartDraw, s.f.)

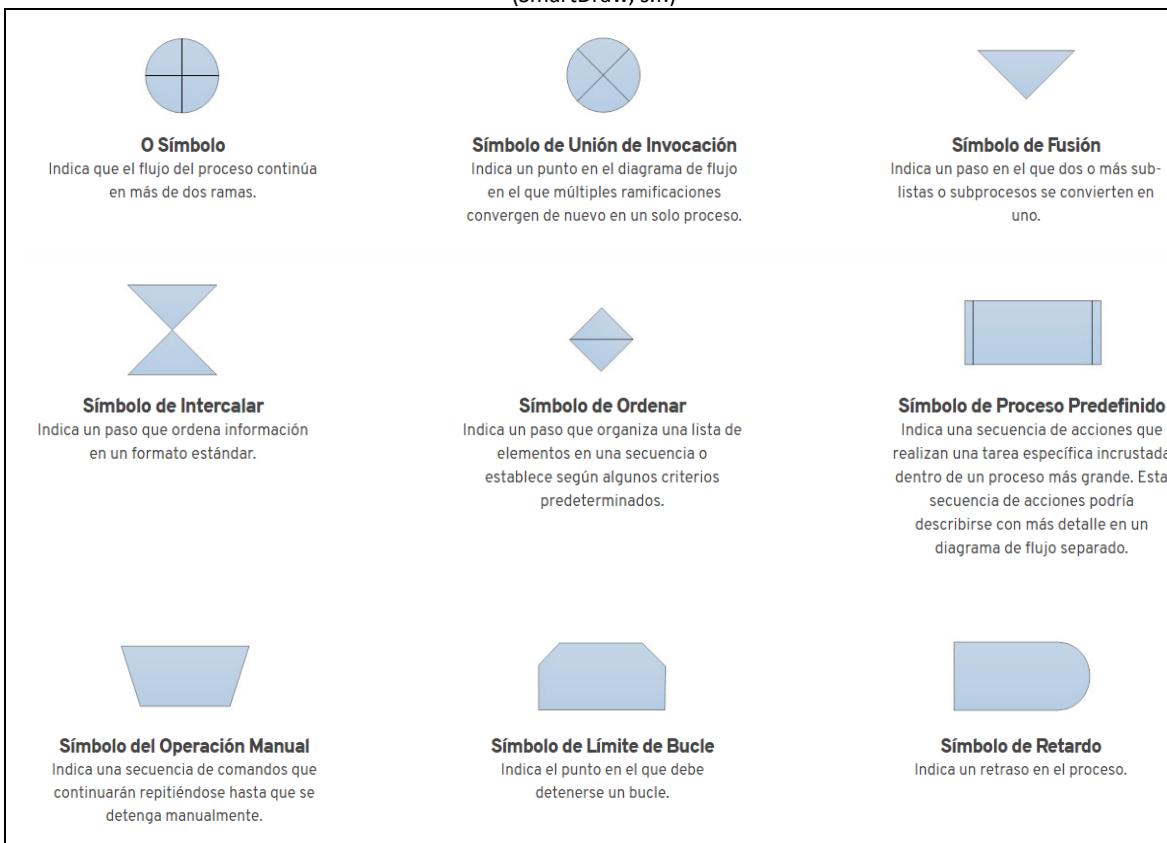




Figura N° 36: Símbolos más utilizados en DF (continuación)
 (SmartDraw, s.f.)

3.6.4 Tipos de diagramas de flujo

Distintos autores describen numerosos tipos de diagramas de flujo en diferentes términos.

- ❖ **Sterneckert**, en su libro en 2003 **Critical Incident Management**, mencionó cuatro tipos de diagramas de flujo populares, enmarcados en el concepto de controles de flujos, en vez del flujo en sí mismo:
 - **Diagramas de flujo de documentos:** *Tienen el propósito de mostrar los controles existentes en el flujo de documentos a través de los componentes de un sistema. se lee de izquierda a derecha y detalla el flujo de documentos a través de numerosas unidades de negocio.*
 - **Diagramas de flujo de datos:** *Indican los controles que rigen los flujos de datos en un sistema, se usan principalmente para mostrar los canales donde se transmiten los datos a través del sistema, en lugar de cómo se controla el flujo.*
 - **Diagramas de flujo de sistemas:** *"Indican el flujo de datos que pasa hacia los componentes principales de un sistema, o a través de ellos, tales como entrada de datos, programas, medios de almacenamiento, procesadores y redes de comunicación".*
 - **Diagramas de flujo de programas:** *Muestran los controles ubicados internamente en un programa dentro de un sistema.*
- ❖ **Veronis**, en su libro en 1978, **Microprocessors: Design and Applications**, describió tres tipos de diagramas de flujo en función del alcance y nivel de detalle:
 - **Diagrama de flujo de sistema:** *Identifica los dispositivos que se emplearán.*
 - **Diagrama de flujo general:** *Vista general.*
 - **Diagrama de flujo detallado:** *Más detalles.*

- ❖ But Fryman, en su libro en 2001 titulado ***Quality and Process Improvement***, distinguió los tipos de muchas maneras, más desde una perspectiva orientada a los negocios que a la informática:
 - ***Diagrama de flujo de decisiones***
 - ***Diagrama de flujo lógico***
 - ***Diagrama de flujo de sistemas***
 - ***Diagrama de flujo de productos***
 - ***Diagrama de flujo de procesos***
- ❖ Otros tipos de diagramas de flujo definidos por otros incluyen:
 - ***Diagrama de carriles:*** También conocido como "diagrama de flujo de carriles": detalla los roles de cada participante en procesos que se realizan entre equipos.
 - ***Diagrama de flujo de trabajo:*** Documenta flujos de trabajo, a menudo involucra tareas, documentos e información en las oficinas.
 - ***Diagrama de cadena de procesos impulsado por eventos (EPC):*** Documenta o planifica un proceso de negocio.
 - ***Diagrama de flujo de lenguaje de descripción y especificación (SDL):*** Realiza una lluvia de ideas sobre los algoritmos informáticos mediante tres componentes básicos: Proceso, bloqueo y definición de sistema.
- ❖ Estos diagramas relacionados también se piensan, a veces, como tipos de diagramas de flujo:
 - ***Diagrama de flujo de datos (DFD):*** Traza el flujo de información de cualquier sistema o proceso.
 - ***Diagrama de flujo de procesos (PFD),*** también conocido como "gráfico de flujo de procesos": ilustra las relaciones entre los principales componentes de una planta industrial.
 - ***Modelo y notación de procesos de negocio (BPMN 2.0):*** Modela los pasos de un proceso de negocio planificado.

3.6.5 Diagrama de flujo de datos (DFD):

Un diagrama de flujo de datos (DFD), es una técnica gráfica que describe el flujo de información y las transformaciones que se aplican a los datos, conforme se mueven de la entrada a la salida, visualiza a un sistema como una red de procesos conectados entre sí. Los DFD son una notación operacional semiformal que ha sido ampliamente adoptada para la especificación de sistemas de información, es independiente del tamaño y de la complejidad del sistema, consiste en un diagrama en forma de red que representa el flujo de datos y las transformaciones que se aplican sobre ellos al moverse desde la entrada hasta la salida del sistema, se apoya en otras dos técnicas: Diccionario de Datos, y Especificaciones de procesos.

Los elementos que componen a un DFD se representan como se indica en la Figura N° 37 y son los siguientes:

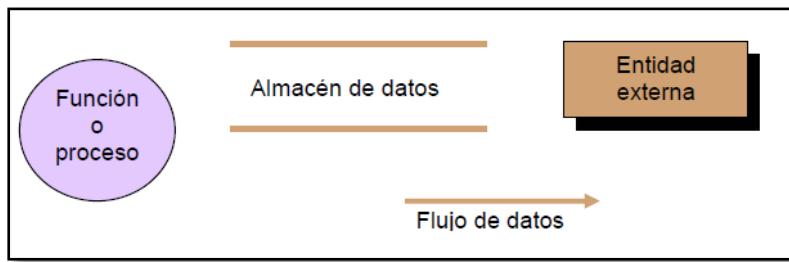


Figura N° 37: Elementos del Diagrama de Flujo de Datos
 (Fuentes, 2011)

3.6.6 Elementos del Diagrama de Flujo de Datos

- **Función o Proceso.** Representa la transformación del flujo de datos. Muestra cómo una o más entradas se transforman en salidas. Su nombre comienza con un verbo y es lo suficientemente largo y claro para que cualquier persona entienda de qué se trata. Dichas funciones van numeradas para diferenciarlas en un mismo nivel mostrando la jerarquía entre los niveles. Se representa por un círculo en cuyo interior está el nombre y el número de la función.
- **Entidad Externa:** Representa el origen o el destino de la información del sistema. Los flujos que parten o llegan a ellas definen el interfaz entre el sistema y el mundo exterior. Su representación gráfica es un rectángulo o cuadrado con el nombre.
- **Almacenamiento.** Son los datos pasivos; generalmente archivos, tablas, etc. Los almacenes de datos representan información del sistema almacenada de forma temporal, por tanto, representan datos en reposo; deben tener un nombre representativo, su representación gráfica son dos líneas paralelas con el nombre en medio, o también se representan con un rectángulo con el nombre adentro.
- **Flujo de dato.** Está representado por una flecha que indica su dirección, va del origen al destino. Los datos siempre van hacia y/o desde una función. Los flujos de datos representan datos en movimiento, los que se mueven hacia y desde almacenes simples no necesitan nombre si transportan toda la información del registro. Cuando se lee o escribe una porción de los elementos se debe especificar el nombre en el flujo. Existen tres tipos: Flujo de entrada, Flujo de salida y Flujo de diálogo. Los procesos pueden introducir o recuperar datos en los almacenes:
 - **Flujo de salida o consulta:** Indica la utilización de la información del almacén con el proceso.



- **Flujo de entrada o actualización:** Indica que el proceso va a alterar la información del almacén.



- **Flujo de diálogo:** Representa como mínimo un flujo de consulta y uno de actualización que no tienen relación directa.



3.7 Modelos entidad-relación

Conocido como Diagramas Entidad- Relación (DER) o Modelo Entidad-Relación (MER), sirven para modelar un almacenamiento de datos. En muchos casos, los datos manipulados por el sistema determinan las acciones que se realizan. Puede ser útil definir los requerimientos concentrándose en los datos en lugar de las funciones. La abstracción de datos es una técnica para describir para qué son los datos, en lugar de cuál es su apariencia como se denominan.

Para describir los datos se utiliza el diccionario de tipo de datos. La idea central es categorizar los datos y agrupar los elementos semejantes.

El Modelo Entidad /Relación fue desarrollado por Chen en 1976. Es un modelo muy utilizado en el campo de diseños de bases de datos. Su principal ventaja es que es traducible casi automáticamente a un esquema de bases de datos bajo modelo relacional.

El aspecto de datos es más estable que el funcional en la mayoría de los sistemas; también es mucho más difícil pensar en modelo de datos. La mayor dificultad se presenta en poder establecer la estructura de los objetos y las relaciones entre los mismos.

3.7.1 Elementos del Modelo Entidad/Relación (MER)

El Modelo Entidad – Relación tiene sus propias estructuras que son los diagramas entidad/Relación (DER).

- **Entidades:** Una entidad es un objeto real o abstracto de interés en una organización acerca del cual se puede y se quiere guardar información. Asociado al concepto de entidad surge el de **Ocurrencia de entidad** que es una realización concreta de la misma. Por ejemplo, si las entidades son **libro, editorial, autor, documento**. Las ocurrencias para la entidad editorial serían: **McGraw-Hill, Addison-Wesley, Alfaomega**.
- **Atributos:** Un atributo es una propiedad o característica asociada a una entidad y común a todas las ocurrencias de esta. Por ejemplo, para la entidad **Alumno** se pueden tener los atributos: Nombre, grupo y calificación, o para la entidad **Curso** se pueden tener los atributos: Unidad, nombre UEA, Carrera.

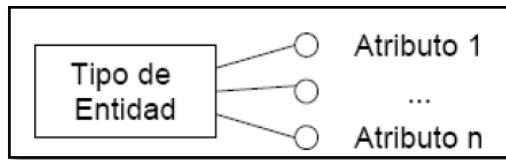


Figura N° 38: *Tipo de Entidad y sus Atributos*
(Fuentes, 2011)

- **Relaciones:** Una relación es una asociación entre entidades. Entre dos entidades pueden existir más de un tipo de relación. Un objeto asociativo es un elemento que sirve para relacionar objetos. Para que exista una instancia de este, deben existir instancias de todos los objetos que relaciona. Asociado al concepto de Relación surge el concepto de ocurrencia de relación que es la asociación concreta de ocurrencias de entidad de las diferentes entidades.

Por ejemplo: “Autor escribe Documento” o “Editorial edita Libro”.

Actividad del Aprendizaje Esperado N° 3**Actividad N° 3**

Desarrolle un algoritmo que permita leer y almacenar el promedio en una variable para luego evaluar la condición del alumno según los siguientes criterios: si el promedio es menor a 4.0 el alumno habrá “Reprobado”, si el promedio es Mayor o igual a 5.5, el alumno se habrá “Eximido” de lo contrario deberá “Rendir Examen”, el algoritmo debe imprimir cuál es el resultado del alumno

A modo de apoyo te entrego el pseudocódigo del algoritmo para que lo puedas interpretar y así poder representar el diagrama de flujo solicitado:

Pseudocódigo	Diagrama de Flujo
<p>1. Inicio</p> <p>2. Inicializar Variable</p> <p><i>PROMEDIO=0</i></p> <p>3. Solicitar introducción de valor</p> <p><i>NOTA</i></p> <p>4. Leer NOTA</p> <p>5. Almacenar en la variable</p> <p><i>PROMEDIO</i></p> <p>6. Si <i>PROMEDIO</i> < 4.0 Entonces</p> <p>7. Escribir <i>PROMEDIO</i></p> <p>“Reprobado”</p> <p>8. Sino</p> <p>9. Si <i>PROMEDIO</i> ≥ 5.5 Entonces</p> <p>10. Escribir <i>PROMEDIO</i> “Eximido”</p> <p>11. Sino</p> <p>12. Escribir <i>PROMEDIO</i> “Rinde Examen”</p> <p>13. Fin_Si</p> <p>14. Fin_Si</p> <p>15. Fin</p>	

II UNIDAD DE APRENDIZAJE: UML PARA EL ANÁLISIS DE SISTEMAS

Aprendizaje Esperado 4

Determinan atributos y métodos de clases, considerando diagramas de clases, organización de clases y técnicas de mejoramiento en diagramas de clases.

4.1. Diagramas de clases

Los Diagramas de Clases pertenecen al grupo de los Diagramas de Estructura, muestran las diferentes clases que componen un sistema y cómo se relacionan unas con otras. Se dice que los diagramas de clases son diagramas “estáticos” porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: Qué clases “conocen” a qué otras clases o qué clases “son parte” de otras clases, pero no muestran los métodos mediante los que se invocan entre ellas. En la Figura N° 39 podemos observar un ejemplo de diagrama de clases. Los diagramas de clases sirven para describir los componentes esenciales de la arquitectura de un sistema. A diferencia de los Diagramas de Flujo de Datos, los Diagramas de Clases muestran relaciones de asociación entre clases y no flujo de datos entre ellas.

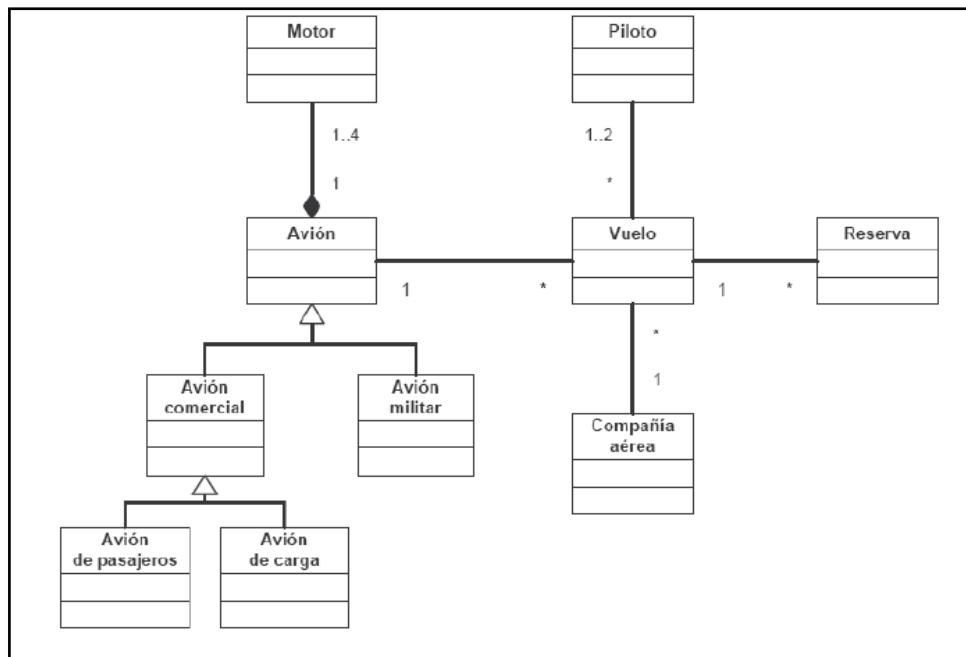


Figura N° 39: Diagrama de Clases de un sistema de Aviación
 (Fuentes, 2011)

4.2. Relaciones entre entidades

Una relación identifica una dependencia. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacia sí misma (menos común, pero existen), este último tipo de

dependencia se denomina dependencia reflexiva. Las relaciones se representan con una línea que une las clases, esta línea variará dependiendo del tipo de relación.

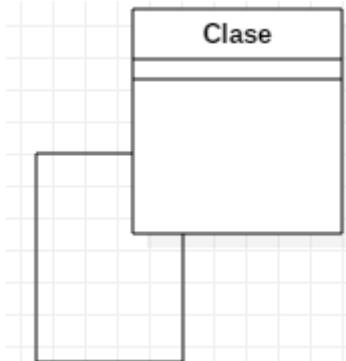


Figura N° 40: Relación Reflexiva
 (Huete-Huerta, s.f.)

Las relaciones en el diagrama de clases tienen varias propiedades, que dependiendo de la profundidad que se quiera dar al diagrama se representarán o no. Estas propiedades son las siguientes:

- **Multiplicidad.** Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango... Se utiliza n o $*$ para identificar un número cualquiera.
- **Nombre de la asociación.** En ocasiones se escribe una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como, por ejemplo: “Una empresa contrata a n empleados”.

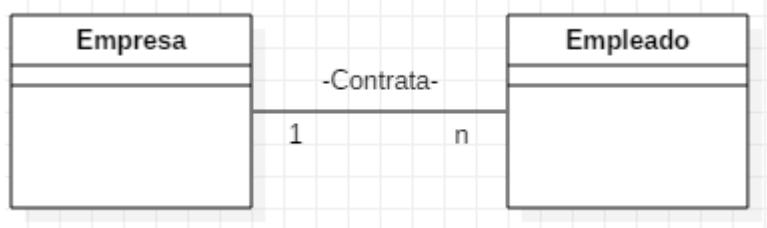


Figura N° 41: Ejemplo de Relación
 (Huete-Huerta, s.f.)

4.2.1. Tipos de relaciones

Un diagrama de clases incluye los siguientes tipos de relaciones:

- Asociación
- Agregación
- Composición
- Dependencia
- Herencia

4.2.1.1. Asociación

Este tipo de relación es el más común y se utiliza para representar dependencia semántica. Se representa con una simple línea continua que une las clases que están incluidas en la asociación.

Un ejemplo de asociación podría ser: “Una mascota pertenece a una persona”.

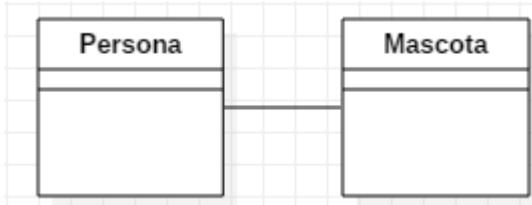


Figura N° 42: Asociación
 (Huete-Huerta, s.f.)

4.2.1.2. Agregación

Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que un objeto es parte de otro, pero aun así debe tener existencia en sí mismo.

Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase, es decir, en la clase que contiene las otras.

Un ejemplo de esta relación podría ser: “Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa”. Como ves, el tornillo podría formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.

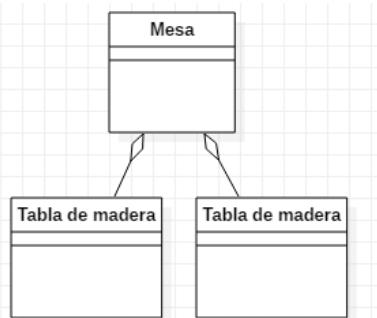


Figura N° 43: Agregación
 (Huete-Huerta, s.f.)

Composición

La composición es similar a la agregación, representa una relación jerárquica entre un objeto y las partes que lo componen, pero de una forma más fuerte. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el primero no existe. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos, sino que dependen del elemento que componen. Además, suelen tener los mismos tiempos de vida. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.

Se representa con una línea continua con un rombo relleno en la clase que es compuesta. Un ejemplo de esta relación sería: "Un vuelo de una compañía aérea está compuesto por pasajeros, que es lo mismo que decir que un pasajero está asignado a un vuelo".

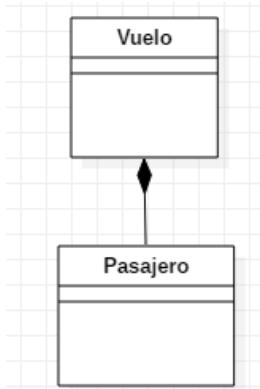


Figura N° 44: Composición
 (Huete-Huerta, s.f.)

4.2.1.3. Dependencia

Se utiliza este tipo de relación para representar que una clase requiere de otra para ofrecer sus funcionalidades. Es muy sencilla y se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.

Un ejemplo de esta relación podría ser la siguiente:

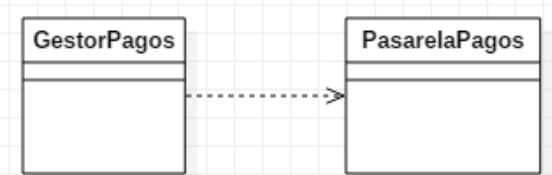


Figura N° 45: Dependencia
 (Huete-Huerta, s.f.)

4.3. Generalización y especialización de clases

Generalización es otro nombre para **herencia**. Se refiere a una relación entre dos clases en donde una Clase “Específica” es una versión especializada de la otra, o Clase “General”.

La generalización es la propiedad que permite compartir información entre dos entidades evitando la redundancia. En el comportamiento de objetos existen con frecuencia propiedades que son comunes en diferentes objetos y esta propiedad se denomina generalización.

Por ejemplo, lavadoras, refrigeradores, microondas, tostadoras, lavavajillas, etc., son todos electrodomésticos (aparatos del hogar).

En el mundo de la orientación a objetos, cada uno de estos aparatos es una subclase de la clase Electrodoméstico y a su vez Electrodoméstico es una superclase de todas las otras clases (lavadoras, microondas, lavavajillas, etc.).

El proceso inverso de la generalización por el cual se definen nuevas clases a partir de otras ya existentes se denomina especialización.

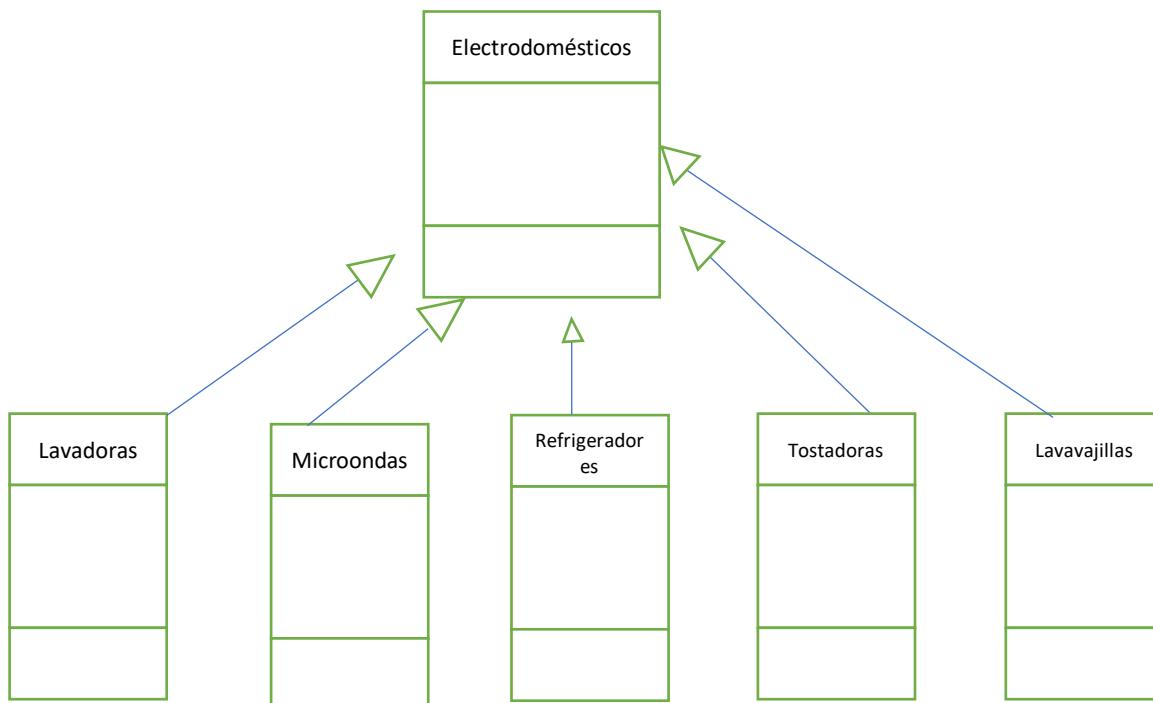


Figura N° 46: Herencia o Generalización de clases
 (Propia, Elaboración, s.f.)

La herencia permite definir nuevas clases a partir de otras clases ya existentes, de modo que presentan las mismas características y comportamiento de estas, así como otras adicionales. Clases diferentes se pueden conectar unas con otras con modo jerárquico.

Las relaciones de generalización y especialización, es tan presentes en nuestras vidas diarias cuando clasificamos animales, objetos o cosas, por eso se utiliza el concepto de clases divididas en subclases. Por ejemplo: La clase animal se divide en anfibios, mamíferos, insectos, pájaros, etc., y la clase vehículo en autos, motos, camiones, buses, etc. El principio de la división o clasificación es que cada subclase comparte características comunes con la clase de la que procede o se deriva.

Los autos, motos, camiones y buses tienen ruedas, motores y carrocerías; son las características que definen a un vehículo. Además de las características comunes con los otros miembros de la clase, cada subclase tiene sus propias características. Por ejemplo, los camiones tienen una cabina independiente de la caja que transporta la carga; los buses tienen un gran número de asientos independientes para los viajeros que ha de transportar, etc.

A la inversa, la clase base es la generalización de la clase derivada. Esto significa que todas las propiedades (atributos y operaciones) de la clase base se heredan por la clase derivada, normalmente suplementada con propiedades adicionales.

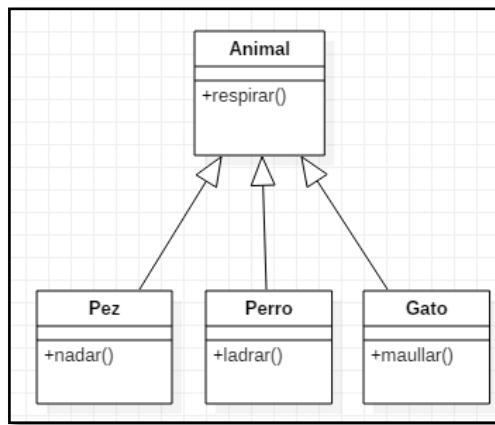


Figura N° 47: Ejemplo de Generalización
 (Huete-Huerta, s.f.)

4.4. Polimorfismo

El polimorfismo significa que una clase (generalmente abstracta) representa un conjunto formado por objetos diferentes, ya que estos son instancias de subclases diferentes. Cuando se llama a un método del mismo nombre, esta diferencia se traduce en comportamientos distintos (excepto en los casos en los que el método es común y las subclases lo han heredado de la superclase).

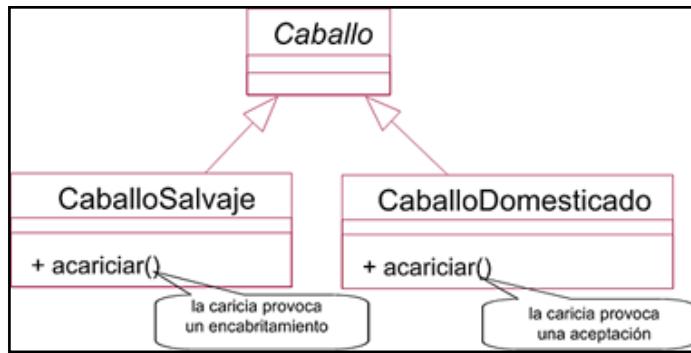


Figura N° 48: Ejemplo de Polimorfismo
 (Eni Ediciones, s.f.)

Es la habilidad que poseen los objetos para reaccionar de modo diferente ante los mismos mensajes. El polimorfismo se refiere a la posibilidad de definir múltiples clases con funcionalidad diferente, pero con métodos o propiedades denominados de forma idéntica, que pueden utilizarse de manera intercambiable mediante código cliente en tiempo de ejecución.

4.4.1. Conceptos relacionados con polimorfismo

- **Sobrecarga (overload):** La sobrecarga representa diferentes maneras de realizar una misma acción. En los programas se usa el mismo nombre en diferentes métodos con diferentes firmas [número, orden y tipo de los parámetros]. El código de programación asociado a cada sobrecarga puede variar.
- **Sobreescritura (override):** Sucede cuando una clase “B” hereda características de una clase “A”, pero la clase “B” redefine las características heredadas de “A”. Propiedades y métodos pueden heredarse de una superclase. Si estas propiedades y métodos son redefinidos en la clase derivada, se dice que han sido “sobrescritos”.

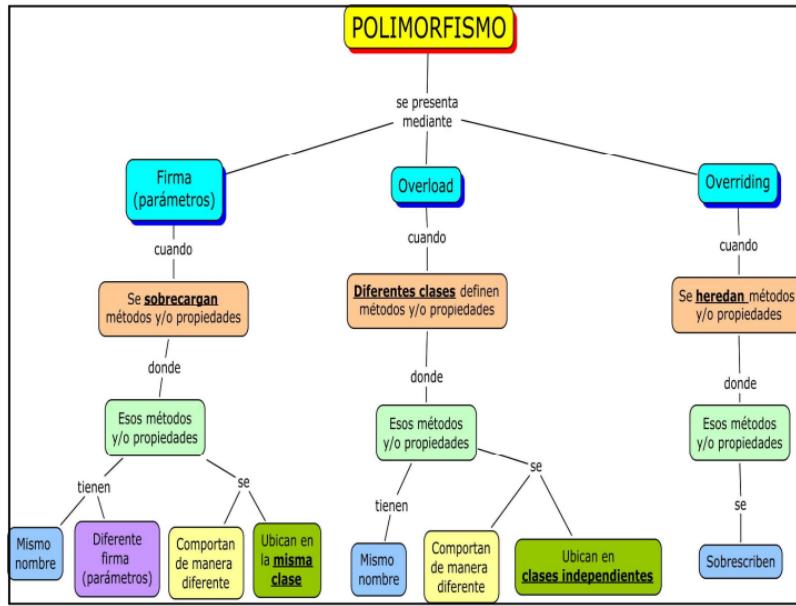


Figura N°49: Representación de Polimorfismo
 (Hernández)

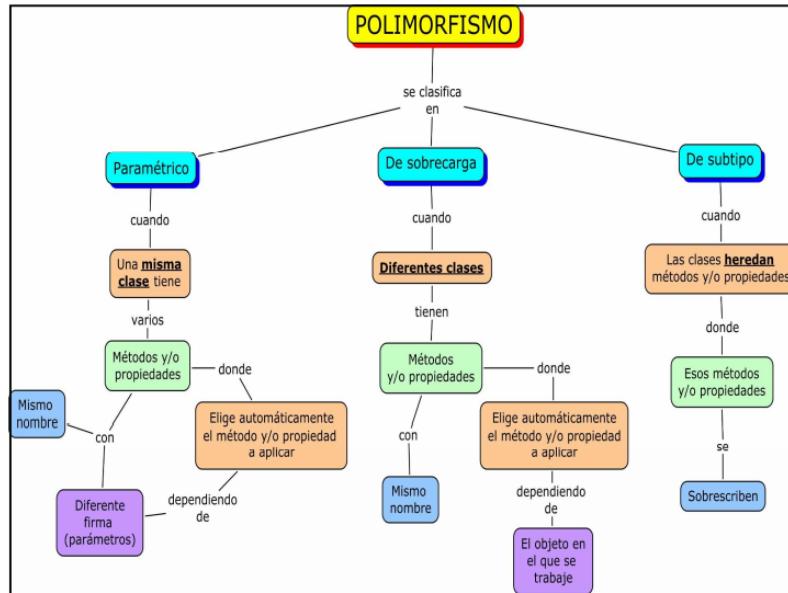


Figura N° 50: Clasificación de Polimorfismo
 (Hernández)

4.5. Clases abstractas

Si examinamos la jerarquía presentada en la Figura N° 51 vemos que en ella existen dos tipos de clases: Las clases que poseen instancias, es decir, las clases **Caballo** y **Perro**, llamadas clases concretas.

Las clases que no poseen directamente instancias, como la clase Animal. En efecto, si bien en el mundo real existen caballos, perros, etc., el concepto de animal propiamente dicho continúa siendo abstracto. No basta para definir completamente un animal. La **clase Animal** se llama clase abstracta. La finalidad de las clases abstractas es poseer subclases concretas y sirven para factorizar atributos y métodos comunes a las subclases.

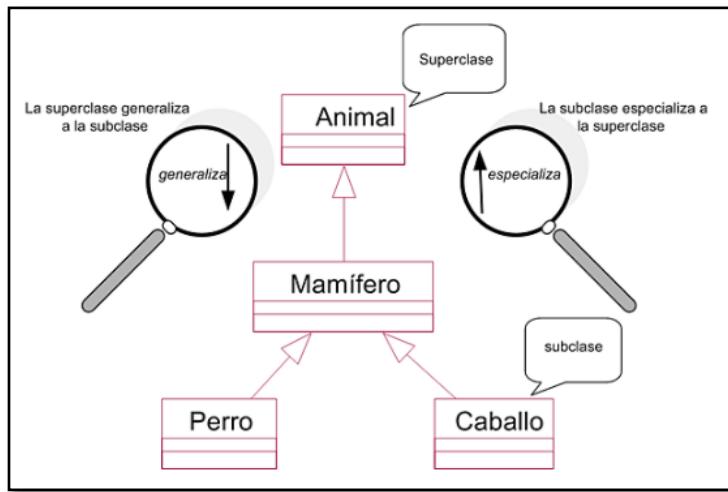


Figura N° 51: Ejemplo jerarquía de Clases
 (Eni Ediciones, s.f.)

4.6. Técnicas de mejoramiento en diagramas de clases

El Diagrama de Clases de UML, si incluyera el concepto de cardinalidad mínima y máxima de los atributos, no solo mejoraría su capacidad expresiva, al permitir especificar restricciones sobre otros aspectos de la realidad que son necesarios transmitir al modelo físico, sino también permitiría su simplificación porque evitaría tener que ascender a un atributo multivalorado a la categoría de entidad, estableciendo una relación m:1 con la entidad que lo poseía; como se sugiere hacer en estos casos.

Asimismo, la admisión de atributos objeto valuados evitaría la representación de relaciones "parte de" o de agregación, cuando esta relación sea inclusiva y estricta; tal como se entiende en OO-Method. Por lo tanto, la representación de atributos objeto valuados, evitaría también la descomposición poco natural de los objetos, aumentando la simplicidad del modelo.

No obstante, al considerar la posibilidad de tener atributos de tipo objeto, se tendría que admitir representar no solo sus atributos, sino también algunas relaciones relevantes. Por lo tanto, también el diagrama de Clases de UML, debería admitir la posibilidad de representar relaciones entre atributos, además de las relaciones entre clases.

Se propone, por otro lado, refinar la relación de dependencia del Diagrama de Clases de UML usada para representar las dependencias cliente/servidor; estableciendo las relaciones entre operaciones, en lugar de relaciones entre clases porque con ello se aumentaría el rigor del modelo.

Para evitar errores cuando se expresa la cardinalidad de las relaciones, se sugiere emplear la notación del Modelo E-R y aunque este cambio es meramente sintáctico, aumentaría la comprensibilidad del modelo.

Por último, se propone reemplazar la notación gráfica de la “visibilidad” (clasificada en pública, privada, protegida o de implementación) de un atributo, considerado aspecto del espacio de la solución, por la unicidad de los atributos que sí es un aspecto que debe vigilarse para evitar duplicidad de información que haga pensar que existen más objetos de los que realmente hay en ese dominio particular.

En primer lugar, la admisión de clases derivadas en el modelo estructural, como ocurre en el Diagrama de Clases de UML, afectaría negativamente, tanto la simplicidad como la minimalidad o suficiencia del modelo. Se propone, por lo tanto, crear una capa o nivel más, en las fases de desarrollo de un sistema informático: El modelo de visión o externo. El modelo de visión, que se propone estaría constituido por la colección de las vistas parciales que tienen los actores del sistema; incluyendo dentro de estos, a los sistemas externos o aplicaciones cliente.

Del modelo de visión, que sería el primero en definirse, deben poderse integrar las vistas externas para formar un único modelo estructural, libre de redundancias o de conceptos que se puedan deducir de otros ya representados; cumpliendo así, con la arquitectura ANSI/SPARC propuesta hace varios años.

El Diagrama de Clases de UML, con todas estas propuestas presentadas para mejorar su calidad, tendría una notación similar a la que aparece en la Figura N° 52.

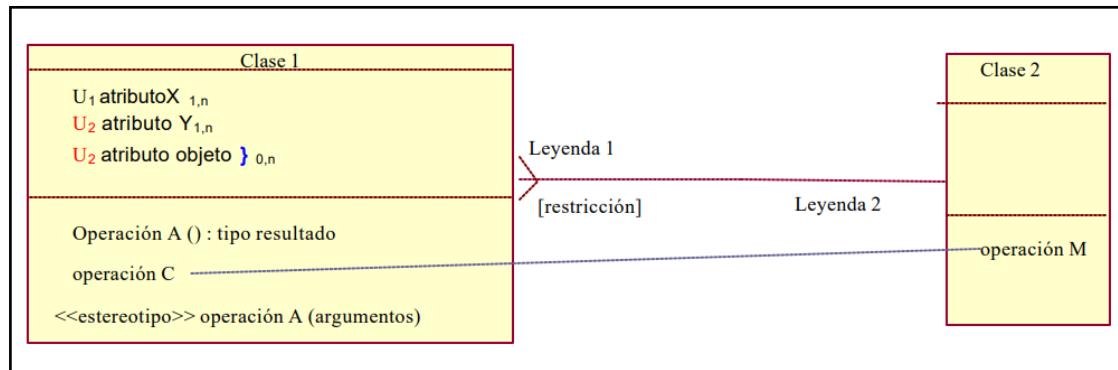


Figura N° 52: Diagrama de Clases de UML, Mejorado
 (Towers, s.f.)

4.7. Atributos y métodos de clases

Un atributo es una propiedad característica de una clase y describe un rango de valores que la propiedad podrá contener en los objetos de una clase. Una clase podrá contener varios o ningún atributo. Por convención, si el atributo consta de una sola palabra se escribe en minúsculas; por otro lado, si el nombre contiene más de una palabra, cada palabra será unida a la anterior y comenzará

con una letra mayúscula, a excepción de la primera palabra que comenzará en minúscula. La lista de nombres de atributos iniciará luego de una línea que la separe del nombre de la clase, como se aprecia en la Figura N° 53.

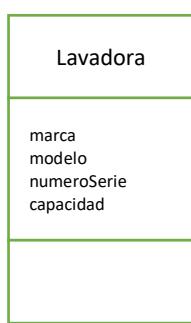


Figura N° 53: Una Clase y sus atributos
 (Propia, Elaboración, s.f.)

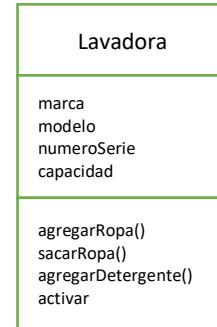


Figura N° 54: Lista de métodos de la Clase Lavadora
 (Propia, Elaboración, s.f.)

Un método es un conjunto de instrucciones que realizan una determinada tarea y son similares a las funciones de los lenguajes estructurados.

Del mismo modo que hay variables de instancia y de clase, también hay métodos de instancia y de clase. En el primer caso, un objeto llama a un método para realizar una determinada tarea, en el segundo, el método se llama desde la propia clase. La lista de operaciones se inicia debajo de una línea que separa a las operaciones de los atributos, como se muestra en la Figura N° 54.

Actividad del Aprendizaje Esperado N° 4

A continuación, represente la jerarquía de clases que presenta los trabajadores de una empresa de Retail, considerando las siguientes indicaciones:

- a) La Clase Trabajador es una clase abstracta con tres atributos (nombre, edad, cargo), un método Tipo contrato, con sus respectivos tipos de datos.
- b) La Clase Administrador no añade nuevos atributos miembro, aunque deberá implementar el método ()”
- c) La clase Vendedor tiene dos nuevos atributos (comisión_%, Jornada)
- d) La Clase Bodeguero tiene dos nuevos atributos (sector, turno)
- e) La Clase Supervisor igual agrega dos atributos (Departamento, Meta)

Para realizar este ejercicio se pide lo siguiente: Crear el diagrama de clases con las clases atributos y métodos que correspondan, representando la jerarquía de clases.

Aprendizaje Esperado 5

Implementan modelos de software orientado a objetos para el análisis de sistemas, considerando uso de lenguaje UML, diagramas de caso de uso, de actividad, de secuencia y de clases.

5.1. Orientación a objetos

Los programas se modelan en torno a objetos que aglutan toda la funcionalidad relacionada con ellos. De este modo en lugar de crear una serie de funciones sin conexión alguna entre ellas, en POO se crean clases, que representan entidades que quieres manejar en tu programa. Por ejemplo, facturas, líneas de factura, clientes, coches... o cualquier entidad que necesites gestionar conceptualmente. En ese sentido, la POO gira más en torno a los datos que en torno a la lógica, que se delega a un segundo plano, ya que forma parte de dichos datos como veremos enseguida.

Una clase por sí sola no sirve de nada, pues no es más que un concepto, sin entidad real. Para poder utilizar una clase en un programa lo que hay que hacer es instanciarla. Instanciar una clase consiste en crear un nuevo objeto concreto de la misma. Es decir, un objeto es ya una entidad concreta que se crea a partir de la plantilla que es la clase. Este nuevo objeto tiene ya "existencia" real, puesto que ocupa memoria y se puede utilizar en el programa. Así un objeto puede ser una persona que se llama Luz Rojas, de 37 años y que en nuestro programa podría hablar, caminar o comer, que son los comportamientos que están definidos en la clase. De este modo, si tenemos que manejar personas podemos ir creándolas a medida que las necesitemos, y actuar sobre ellas individualmente. Cada una tiene sus propios datos y sus propias acciones.

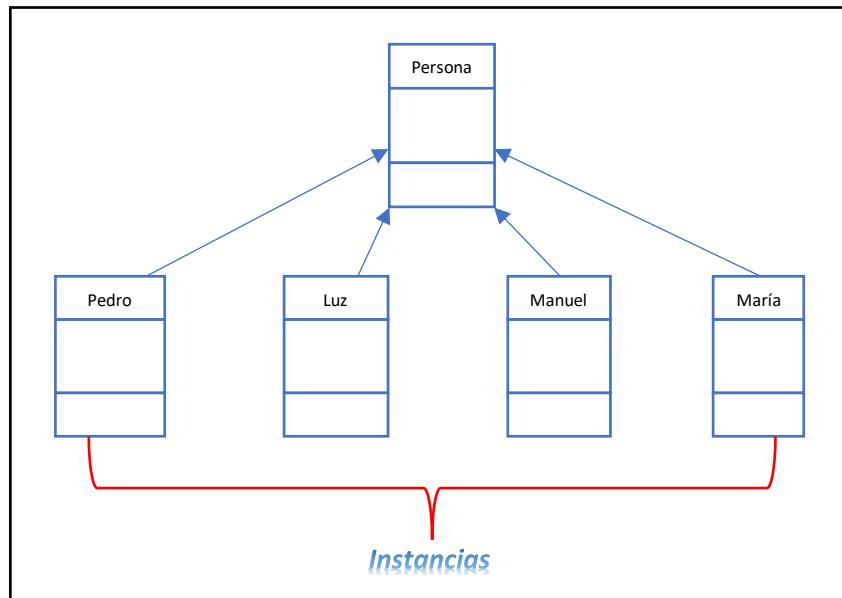


Figura N° 55: Instancias de Clases
 (Propia, Elaboración, s.f.)

5.1.1. Principios de la Programación Orientada a Objetos

- **Encapsulación:** La encapsulación es la característica que permite que todo lo referente a un objeto quede aislado dentro de este. Es decir, que todos los datos referentes a un objeto queden "encerrados" dentro de este y solo se puede acceder a ellos a través de los miembros que la clase proporcione (propiedades y métodos).

Por ejemplo, en el caso de las personas que estábamos viendo, toda la información sobre estas (nombre, apellidos, edad... y cualquier otro dato interno que se utilice y que no necesariamente se ve desde el exterior del objeto) está circunscrito al ámbito de dicha persona.

Así, internamente tenemos un dato que es el nombre de la persona y accedemos a él a través de la propiedad pública Nombre que define la clase que representa a las personas. De este modo damos acceso solo a lo que nos interese y del modo que nos interese.

- **Abstracción:** El principio de abstracción lo que implica es que la clase debe representar las características de la entidad hacia el mundo exterior, pero ocultando la complejidad que llevan aparejada. O sea, nos abstraer de la complejidad que haya dentro dándonos una serie de atributos y comportamientos (propiedades y funciones) que podemos usar sin preocuparnos de qué pasa por dentro cuando lo hagamos.

En nuestro ejemplo de las personas, puede ser que tengamos un dato interno que llamamos energía y que nunca es accesible directamente desde fuera. Sin embargo, cada vez que la persona anda (o corre, si tuviésemos un método para ello) gasta energía y el valor de este dato disminuye. Y cuando la persona come, el valor sube en función de lo que haya comido.

- **Herencia:** Desde el punto de vista de la genética, cuando una persona obtiene de sus padres ciertos rasgos (el color de los ojos o de la piel, una enfermedad genética, etc....) se dice que los hereda. Del mismo modo en POO cuando una clase hereda de otra obtiene todos los rasgos que tuviese la primera. Dado que una clase es un patrón que define cómo es y cómo se comporta una cierta entidad, una clase que hereda de otra obtiene todos los rasgos de la primera y añade otros nuevos y además también puede modificar algunos de los que ha heredado.
- **Polimorfismo:** La palabra polimorfismo viene del griego "polys" (muchos) y "morpho" (forma), y quiere decir "cualidad de tener muchas formas". En POO, el concepto de polimorfismo se refiere al hecho de que varios objetos de diferentes clases, pero con una base común, se pueden usar de manera indistinta, sin tener que saber de qué clase exacta son para poder hacerlo.

5.2. Interpretación de sistemas de software

Para construir algo primero debe entenderse lo que debe ser ese algo. El proceso de entender y documentar una aplicación software se llama “Análisis de requisitos”. En general, los requisitos expresan qué se supone debe hacer una aplicación y no intentan expresar cómo logra estas funciones.

El análisis inicial de un sistema debe tratar de descubrir los requerimientos del producto final que se desarrolla en detalle.

Unos de los principales objetivos de UML es hacer que este análisis sea lo suficientemente intuitivo para que los clientes y expertos en el dominio que solicitan el producto puedan comprenderlo, y lo suficientemente formal y riguroso para que se establezca una formulación no ambigua que pueda ser utilizada por los técnicos que la desarrollan.

Los aspectos básicos que deben tratarse en esta fase son:

- Determinar los paquetes de funcionalidad y de la calidad de servicio del producto, formulados de una forma independiente de su implementación, y refinar y detallar estas especificaciones hasta que den lugar a una especificación no ambigua del producto que se desarrolla.
- Identificar los actores externos al sistema que interactúan con la aplicación de forma relevante.
- Identificar la semántica y las características de los mensajes que intercambian los actores con el sistema que se desarrolla.
- Refinar los protocolos de interacción que usan los actores para llevar a cabo las diferentes transacciones que se pueden realizar con el sistema.

5.3. Lenguaje UML¹⁷

El UML es una de la herramienta más emocionante en el mundo actual del desarrollo del sistema. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas. La comunicación es de suma importancia ya que los analistas de sistemas intentan evaluar los requerimientos de sus clientes, generar un análisis de requerimientos en algún tipo de notación que ellos mismos comprendieran, aunque el cliente no lo comprendiera, para dar tal análisis a uno o varios programadores y esperar que el producto final cumpliese con lo que el cliente deseaba.

¹⁷ Parrafo extraido de Guille, E. (n.d.). Ingeniería Systems. Ingenieriasystems.Com. Retrieved March 23, 2021, from <http://www.ingenieriasystems.com/2013/10/Introduccion-al-UML-Parte-1-de-3.html>

Dado que el desarrollo del sistema es una actividad humana, hay muchas posibilidades de cometer errores en cualquier etapa del proceso, por ejemplo, el analista pudo haber malentendido al cliente, es decir, probablemente produjo un documento que el cliente no pudo comprender. Tal vez ese documento tampoco fue comprendido por los programadores quienes por ende pudieran generar un programa difícil de utilizar y no generar una solución al problema original del cliente.

El UML es la creación de Grady Booch, James Rumbaugh e Ivar Jacobson lanzada al mercado en 1997 y respaldada por OMG generando nuevas versiones y manteniendo una evolución continua.

El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. A continuación, se listarán los diagramas más comunes del UML.

- Diagrama de Clases
- Diagrama de Objetos
- Diagrama de Casos de Uso
- Diagrama de Estados
- Diagrama de Secuencias
- Diagrama de Actividades
- Diagrama de Colaboración
- Diagrama de Componentes
- Diagrama de Distribución

5.4. Implementación de modelos de software orientado a objetos

Un diagrama de implementación UML se enfoca en la estructura de un sistema de software y es útil para mostrar la distribución física de un sistema de software entre plataformas de hardware y entornos de ejecución.

La Figura N° 56 muestra el diagrama de implementación para tal paquete. En ese diagrama, los componentes de hardware se dibujan en cajas marcadas con “<<dispositivo>>”. Las rutas de comunicación entre componentes de hardware se dibujan con líneas con etiquetas opcionales. En la Figura N° 56, las rutas se etiquetan con el protocolo de comunicación y con el tipo de red utilizado para conectar los dispositivos. Cada nodo que hay en un diagrama de implementación también puede anotarse con detalles del dispositivo. Por ejemplo, en la Figura N° 56 se ilustra el navegador cliente para mostrar que contiene un artefacto que consiste en el software del navegador web. Un artefacto por lo general es un archivo que contiene software que corre en un dispositivo. También puede especificar valores etiquetados, como se muestra en la Figura N° 56 en el nodo del servidor web. Dichos valores definen al proveedor del servidor web y al sistema operativo que usa el servidor.

Los diagramas de implementación también pueden mostrar nodos de entorno de ejecución, que se dibujan como cajas que contienen la etiqueta “<<entorno de ejecución>>”. Dichos nodos representan sistemas, como sistemas operativos, que pueden albergar otro software.

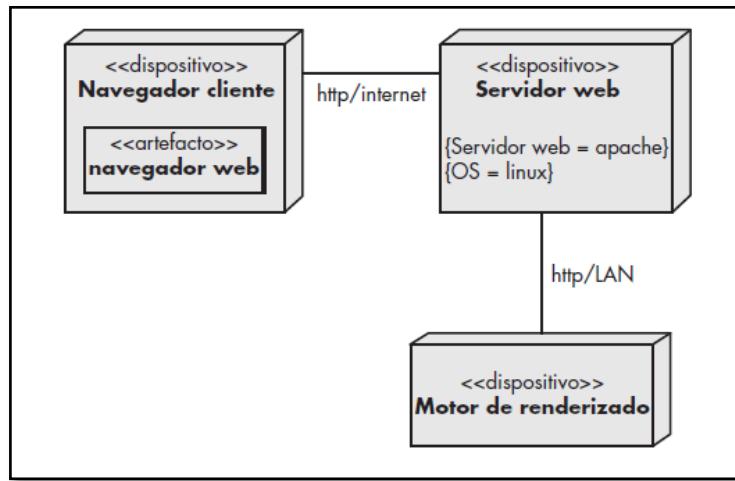


Figura N° 56: Diagrama de Implementación
 (Pressman, 2010)

5.5. Diagramas de casos de uso

Los Diagramas de Casos de Uso pertenecen al grupo de los Diagramas de Comportamiento. Un caso de uso es una interacción entre el sistema y una entidad externa. En su forma más simple, un caso de uso identifica el tipo de interacción y los actores involucrados. Primero se identifican los eventos externos a los que el sistema en desarrollo debe responder y, en segundo lugar, se relacionan estos eventos con los actores y los casos de uso, Figura N° 57.

Los Diagramas de Casos de Uso especifican un sistema en términos de su funcionalidad. A diferencia de las metodologías estructuradas, los diagramas de casos de uso no se descomponen en funciones de programación. Los diagramas de casos de uso son responsables principalmente de documentar los macro requisitos del sistema. Piense en los diagramas de casos de uso como la lista de las capacidades que debe proporcionar el sistema.

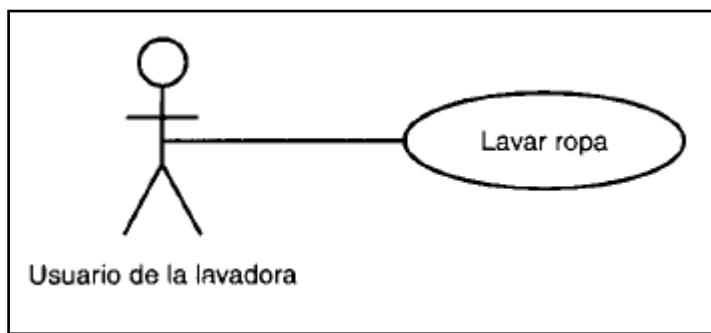


Figura N° 57: Diagrama de Casos de Uso UML
 (Kendall & Kendall, 2011)

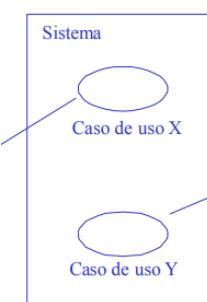
5.7. Modelamiento de casos de uso

Un caso de uso es una técnica de modelado usada para describir lo que debería hacer un sistema nuevo o lo que hace un sistema que ya existe. Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el punto de vista de un usuario, permiten definir los límites del sistema y las relaciones entre el sistema y el entorno.

- Los componentes primarios de un modelo de casos de uso (case-use model) son los casos de uso (use cases), los actores y el sistema modelado.



Caso de Uso: Los casos de uso son descripciones funcionales del sistema; describen cómo los actores pueden usar un sistema.



Límite del Sistema: Los límites del sistema se definen por la funcionalidad que se maneja en el sistema, ser representan con un rectángulo que abarca los casos de uso correspondientes a esos límites.

Funcionalidad: La funcionalidad se representa mediante diversos casos de uso, (X – Y) especificando cada uno una funcionalidad completa (desde su inicio por parte de un actor externo hasta que haya realizado la funcionalidad requerida). Un caso de uso siempre debe devolver algún valor a un actor, siendo el valor cualquier cosa que el actor desee del sistema.



Actor A

Actor: El actor es una entidad externa que tiene interés en interactuar con el sistema. A menudo, es una persona que usa el sistema, pero también puede ser otro sistema o alguna clase de dispositivo hardware que necesita interactuar con el sistema.

En el modelado de casos de uso, el sistema se observa como una caja negra que proporciona casos de uso. Cómo lo haga el sistema, cómo se implementen los casos de uso y cómo trabajen internamente no importa.

- **Clases e Interacciones:** Las clases e interacciones que implementan los casos de uso del sistema. Las interacciones se expresan en diagramas de secuencia, colaboración y actividad, así, hay un enlace entre las vistas funcional y dinámica del sistema. Las clases en la implementación de los casos de uso se modelan y se describen en diagramas de clases y de estados.

5.6.1. Relaciones entre casos de uso

- **Relación de inclusión (include):** Un caso de uso incluye el comportamiento completo de un caso de uso general. Por lo tanto, el caso de uso incluido en el caso de uso base, es parte fundamental del proceso por eso es necesario incluirlo, además Permite la composición jerárquica de casos de uso, así como la reutilización entre casos de uso (Figura N° 59).

- **Relación de extensión (extend):** Un caso de uso añade acciones, que pueden ser opcionales, al comportamiento de un caso de uso general. El caso de uso extendido puede incluir comportamiento del caso de uso que se extiende, aunque no tiene que incluir todo el comportamiento (Figura N° 58).

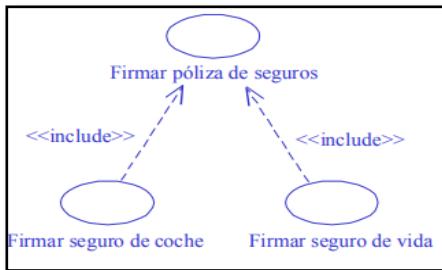


Figura N° 59: Relación Include
 (Mediavilla)

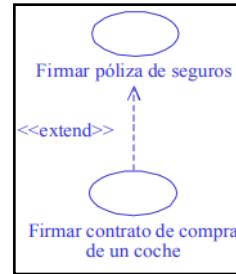


Figura N° 58: Relación Extend
 (Mediavilla)

5.8. Diagramas de actividad

Un diagrama de actividad UML muestra el comportamiento dinámico de un sistema o de parte de un sistema a través del flujo de control entre acciones que realiza el sistema. Es similar a un diagrama de flujo, excepto porque un diagrama de actividad puede mostrar flujos concurrentes.

El componente principal de un diagrama de actividad es un nodo acción, representado mediante un rectángulo redondeado, que corresponde a una tarea realizada por el sistema de software. Las flechas desde un nodo acción hasta otro indican el flujo de control; es decir, una flecha entre dos nodos acción significa que, después de completar la primera acción, comienza la segunda acción. Un punto negro sólido forma el nodo inicial que indica el punto de inicio de la actividad. Un punto negro rodeado por un círculo negro es el nodo final que indica el fin de la actividad (Figura N° 60).

Los diagramas de actividades se usan para analizar los procesos y, si es necesario, volver a realizar la ingeniería de los procesos.

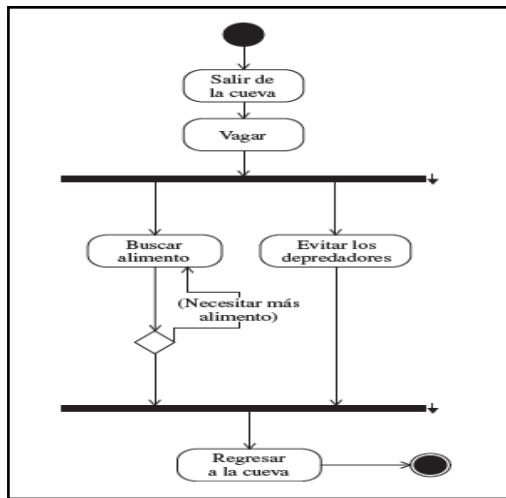


Figura N° 60: Un diagrama de actividades en el que se muestra la manera en que actor camina
 (Pressman, 2010)

Una de las cosas que NO dice el diagrama de actividad de la Figura N° 60 es quién o qué hace cada una de las acciones. Con frecuencia, no importa la división exacta de la mano de obra. Pero si quiere indicar cómo se dividen las acciones entre los participantes, puede decorar el diagrama de actividad con “canales”, como se muestra en la Figura N° 61. Los canales, se forman dividiendo el diagrama en tiras o “carriles” (como si fuera una piscina con carriles de natación), cada uno de los cuales corresponde a uno de los participantes.

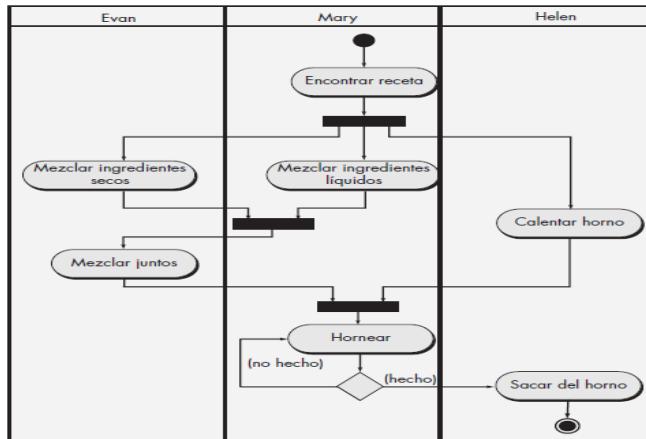


Figura N° 61: Diagrama de Actividad con canales
 (Pressman, 2010)

5.9. Diagramas de secuencia

Los Diagramas de secuencia pertenecen al grupo de los Diagramas de Interacción, sirven para describir los aspectos dinámicos del sistema, mostrando el flujo de eventos entre objetos en el tiempo. Muestran el intercambio de mensajes (es decir la forma en que se invocan) en un momento dado. Ponen especial énfasis en el orden y el momento en que se envían los mensajes a los objetos.

Los objetos están representados por líneas intermitentes verticales, con el nombre del objeto en la parte más alta. El eje de tiempo también es vertical, incrementándose hacia abajo, de forma que los mensajes son enviados de un objeto a otro en forma de flechas con los nombres de la operación y los parámetros. En la Figura N° 62 podemos observar un ejemplo.

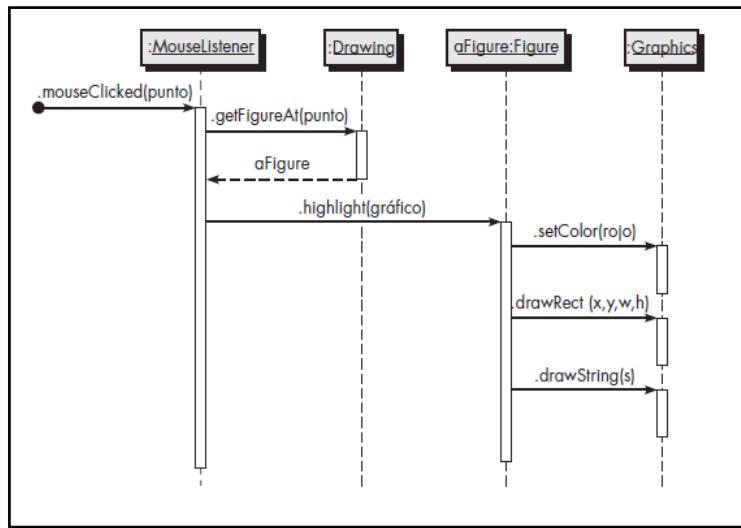


Figura N° 62: Diagrama de Secuencia
 (Pressman, 2010)

5.10. Diagramas de clases

Recordando el tema tratado más atrás **Diagramas de clases** 76; los diagramas de clases se usan para mostrar las clases de un sistema y las relaciones entre ellas (Figura N° 63). Una sola clase puede mostrarse en más de un diagrama de clases y no es necesario mostrar todas las clases en un solo diagrama monolítico de clases. El mayor valor es mostrar las clases y sus relaciones desde varias perspectivas, de una manera que ayudará a transmitir la comprensión más útil.

Los diagramas de clases muestran una vista estática del sistema; no describen los comportamientos o cómo interactúan los ejemplos de las clases. Para describir los comportamientos y las interacciones entre los objetos de un sistema, podemos revisar los diagramas de interacción.

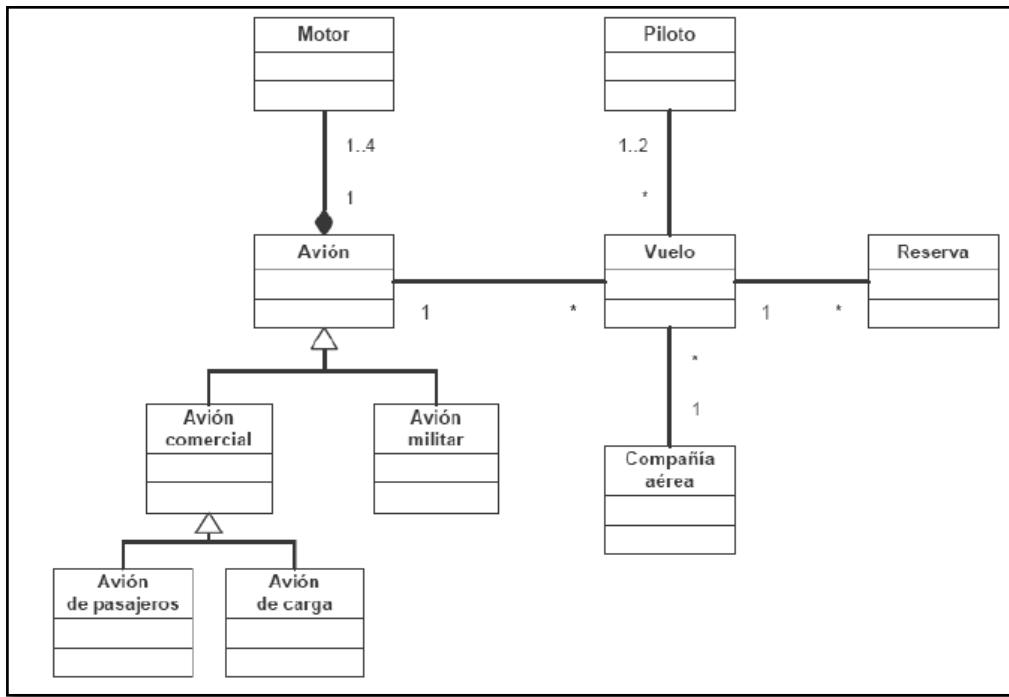


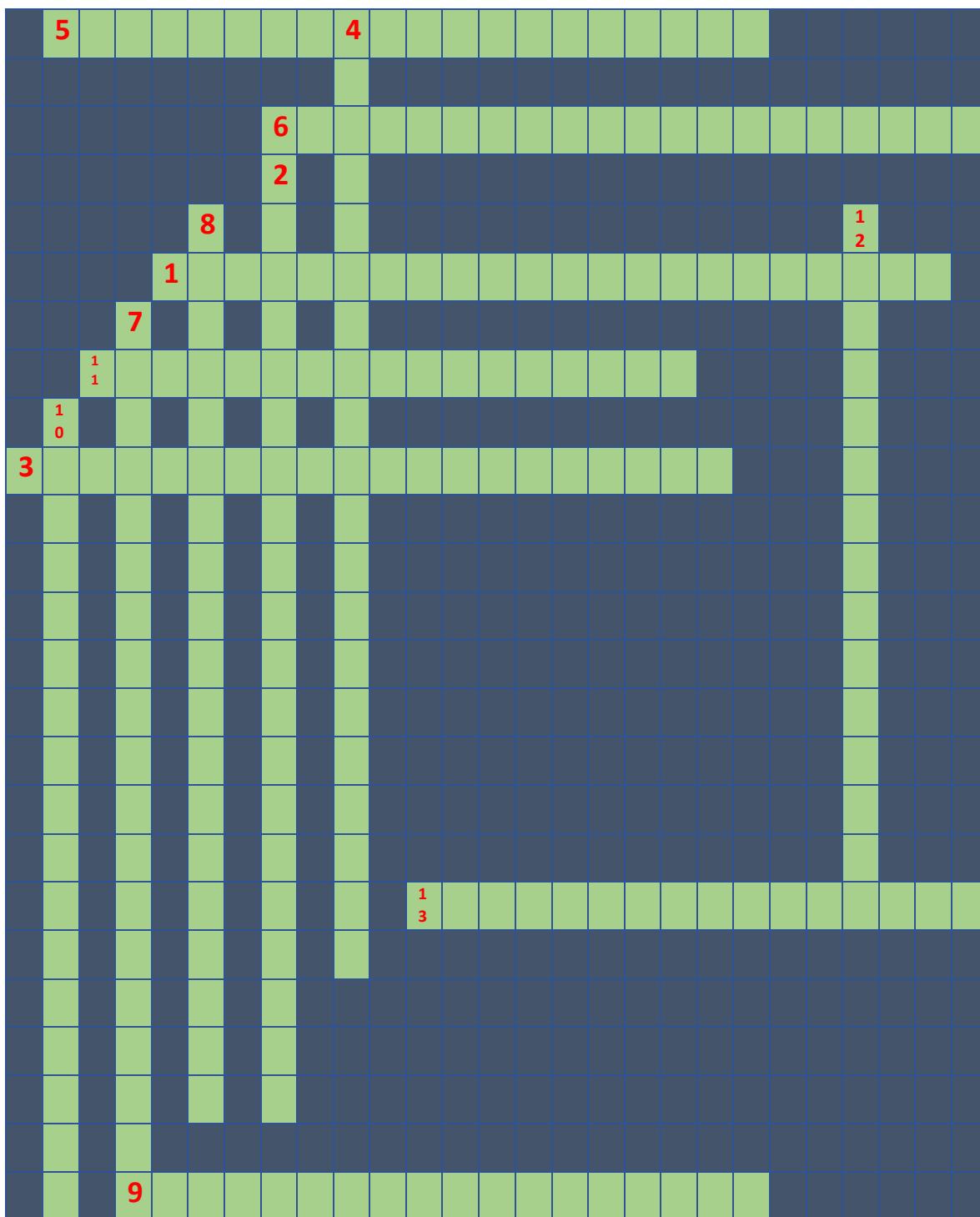
Figura N° 63: Diagrama de Clases de un sistema de aviación
(Fuentes, 2011)

Actividad del Aprendizaje Esperado N° 5**Resuelva el siguiente Crucigrama basado en el Aprendizaje Esperado N° 5:****Verticales**

- 2:** Describe los elementos que componen un sistema. Debe detallar los elementos o componentes, las interacciones y relaciones, así como las interfaces públicas.
- 4:** Muestra la arquitectura de ejecución de un sistema. Incluye nodos, entornos de hardware y software.
- 7:** Representa los procesos de negocio o la lógica de un sistema complejo. Incluye, opcionalmente, el flujo de datos. El nivel de abstracción suele ser bastante alto, pero pueden realizarse diagramas de actividad exploratorios cuando la lógica que se trata es compleja.
- 8:** Muestra la secuencia de la lógica, el orden en que se suceden los mensajes. Importante, especialmente cuando se trabaja en ambientes altamente compartidos.
- 10:** Describe los estados de un objeto, así como la transición entre estados.
- 12:** Muestra una colección de clases, sus tipos, sus contenidos y sus relaciones. Importantísimo representa el modelo de datos, y en consecuencia su persistencia en alguna forma de almacenamiento.

Horizontales

- 1:** Muestra las relaciones entre instancias de las clases y el flujo de mensajes entre ellas.
- 3:** Muestra un panorama general del flujo de control dentro del sistema o proceso de negocio.
- 5:** Muestra casos de uso individuales, actores y las relaciones entre ellos. El Proceso Unificado dice está dirigido por los casos de uso, esto significa que este diagrama (en el nivel de abstracción que sea) es la base del lenguaje de modelado y representación.
- 6:** Muestra la estructura interna de una clase, componente o caso de uso. Especialmente debe indicar los puntos de interacción con otras partes del sistema.
- 9:** Describe como los elementos del modelo se organizan en \"paquetes\", debe indicar la dependencia entre paquetes.
- 11:** Describe los objetos y sus relaciones en algún momento. Generalmente se usa en casos especiales para diagramas de clase o de comunicaciones.
- 13:** Muestra el cambio de estado de un objeto a través del tiempo en respuesta a eventos externos.



Aprendizaje Esperado 6

Modelan sistemas complejos, considerando distintos modelos, capacidades de hardware y diagramas avanzados.

6.1. Modelos de casos de uso y clases

Un modelo de caso de uso es un modelo de las funciones previstas del sistema y su entorno y sirve como un contrato entre el cliente y los desarrolladores. Los casos de uso sirven como hebra de unión a lo largo del desarrollo del sistema. El mismo modelo de caso de uso es el resultado de la disciplina de Requisitos y se utiliza como entrada para disciplinas de Prueba, Diseño y Análisis.

En el diagrama que se incluye más abajo se muestra una parte de un modelo de caso de uso para el Sistema de máquinas de reciclaje (Figura N° 64). Diagrama de caso de uso que muestra un ejemplo de un modelo de uso con actores y casos de uso.

Hay muchos modos de modelar un sistema, y cada uno de ellos puede servir para un objetivo diferente. Sin embargo, el objetivo más importante de un modelo de caso de uso es comunicar el comportamiento del sistema al cliente o usuario. Por lo tanto, el modelo debe ser fácil de comprender.

Los usuarios y cualquier otro sistema que pueda interactuar con el sistema son los actores. Puesto que representan usuarios del sistema, los actores ayudan a delimitar el sistema y a facilitar una imagen más clara de lo que se supone que se debe hacer. Los casos de uso se desarrollan en base a las necesidades de los actores, garantizando así que el sistema resulta ser lo que los usuarios esperaban.

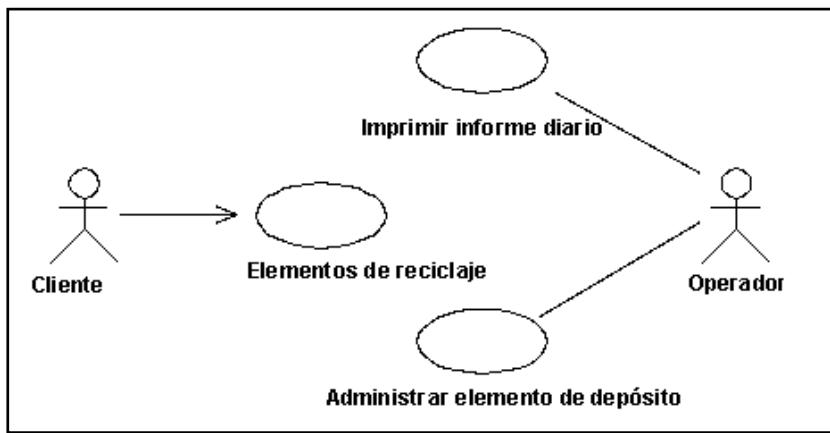


Figura N° 64: Modelo de Casos de Uso
(Pressman, 2010)

De todas maneras el modelo de casos de uso lo hemos detallado más atrás **Diagramas de casos de uso** en la página 72, también hemos desarrollado el modelo de clases más atrás **Diagramas de clases** en la página 56 .

6.2. Diagramas de paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Cuando se usan para representaciones, los diagramas de paquete de los elementos de clase se usan para proveer una visualización de los espacios de nombres. Los usos más comunes para los diagramas de paquete son para organizar diagramas de casos de uso y diagramas de clase, a pesar de que el uso de los diagramas de paquete no es limitado a estos elementos UML.

Los paquetes se pueden construir para representar relaciones tanto físicas como lógicas. Cuando se elige incluir las clases a los paquetes específicos, es útil asignar las clases con la misma jerarquía de herencia a los paquetes, las clases que están relacionadas a través de la composición y las clases que colaboran que también tienen un fuerte argumento para ser incluidas en el mismo paquete.

Los paquetes se representan en UML como carpetas y contienen los elementos que comparten un espacio de nombre; todos los elementos dentro de un paquete deben tener un identificador único. El paquete debe mostrar el nombre del paquete y puede opcionalmente mostrar los elementos dentro del paquete en compartimientos extras.

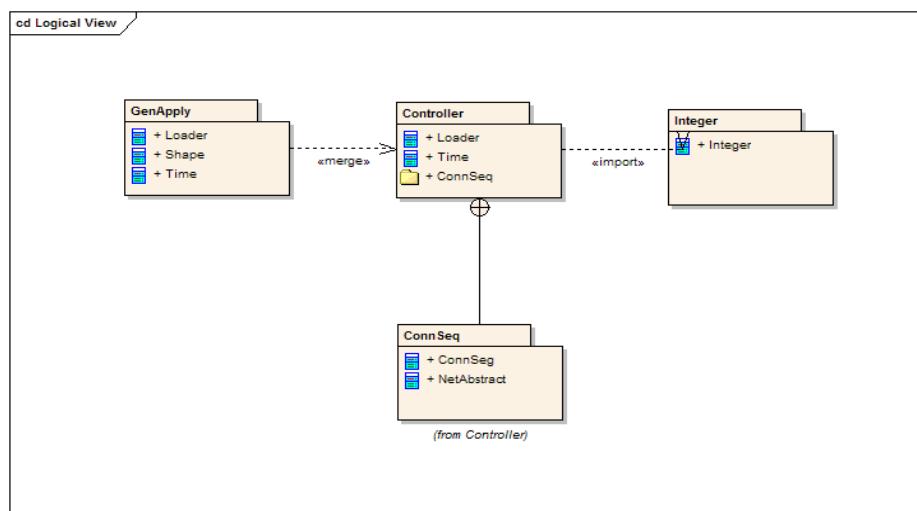


Figura N° 65: Diagrama de Paquetes
 (Sparx Systems, s.f.)

6.3. Interpretación de modelos físicos de sistemas

El modelo físico en UML describe los componentes, de hardware y de software, que se desplegarán en el ambiente seleccionado. Describe elementos tales como plataformas de hardware, denominadas nodos en UML, conectividad de redes, componentes de software, procesadores, sistemas operativos y herramientas de terceras partes.

Los diagramas de despliegue son los complementos de los diagramas de componentes que, unidos, proveen la vista de implementación del sistema.

El Modelo Físico/de Despliegue provee un modelo detallado de la forma en la que los componentes se desplegarán a lo largo de la infraestructura del sistema. Detalla las capacidades de red, las especificaciones del servidor, los requisitos de hardware y otra información relacionada al despliegue del sistema propuesto.

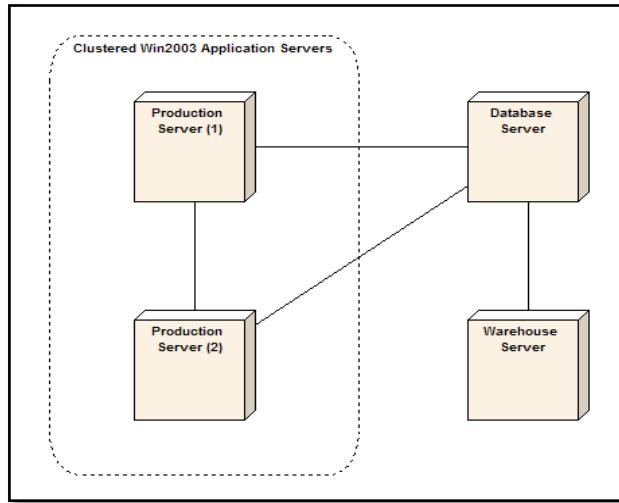


Figura N° 66: Vista de Despliegue
(Sparx Systems, s.f.)

MF01: Modelo Físico

El modelo físico muestra dónde y cómo se desplegarán los componentes. Es un mapa específico de la instalación física del sistema. Un diagrama de despliegue ilustra el despliegue físico del sistema en un ambiente de producción (o prueba). Muestra dónde se ubicarán los componentes, en qué servidores, máquinas o hardware. Puede ilustrar vínculos de red, ancho de banda de LAN, etc.

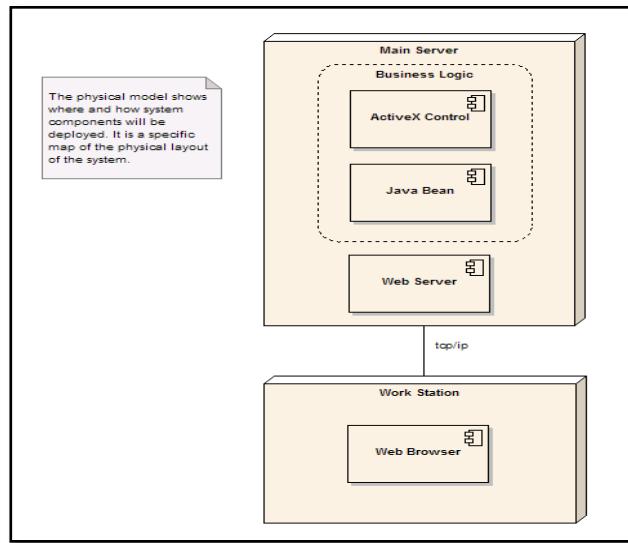


Figura N° 67: MF01 Modelo Físico
 (Sparx Systems, s.f.)

Se utiliza un nodo para identificar cualquier servidor, terminal de trabajo u otro hardware host que se utiliza para desplegar componentes en el ambiente de producción. Usted también puede especificar los vínculos entre los nodos y asignarles estereotipos (tales como TCP/IP) y requisitos. Los nodos también pueden tener documentadas características de performance, estándares mínimos de hardware, niveles de sistema operativo, etc. La pantalla de abajo ilustra las propiedades comunes que puede establecer para un nodo.

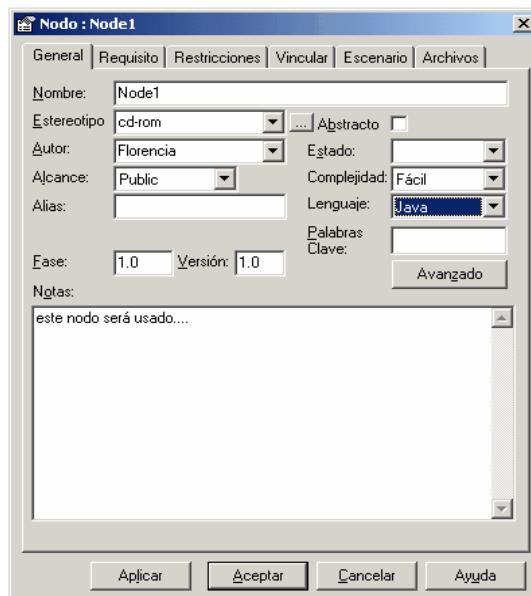


Figura N° 68: Nodo y sus propiedades
 (Sparx Systems, s.f.)

6.4. Diagramas de componentes

¿Qué es un Componente?

Un componente de software es una parte física de un sistema, y se encuentra en el computador, no en la mente del análisis. Entonces, **¿qué puede tomarse como componente?** una tabla, archivos de datos, ejecutables, biblioteca de vínculos dinámicos, documentos y cosas por el estilo.

6.4.1. Tipos de Componentes

- **Componentes de Distribución:** Son los que conforman el fundamento de los sistemas ejecutables, por ejemplo: DLL, Ejecutables, Controles ActiveX y Java Beans.
- **Componentes para trabajar en el producto:** A partir de los cuales se han creado los componentes de distribución como archivos bases de datos y de código.
- **Componentes de Ejecución:** Creados como resultado de un sistema en ejecución.

Un diagrama de componentes contiene, componentes, interfaces y relaciones. También pueden aparecer otros tipos de símbolo que ya haya visto. El símbolo principal de un diagrama de componentes es un rectángulo que tiene otros dos rectángulos más pequeños sobrepuertos en su lado izquierdo un la Figura N° 69 muestra este Símbolo. Debe colocar el nombre del componente dentro del símbolo. El nombre es una cadena.

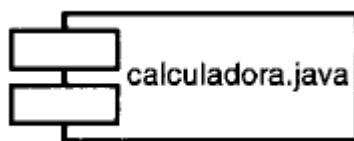


Figura N° 69: Símbolo que representa un componente
(Kendall & Kendall, 2011)

La Figura N° 70 le muestra que, si el componente es miembro de un paquete, puede utilizar el nombre del paquete como prefijo para el nombre del componente. También puede agregar información que muestra algún detalle del componente. El símbolo de la derecha de la Figura N° 70 muestra las clases que implementan un componente en particular.

La Figura N° 71 muestra otra forma de hacer estos, aunque esta técnica por lo general desordena el diagrama.

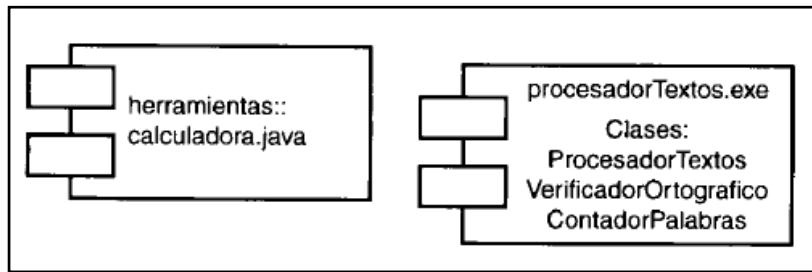
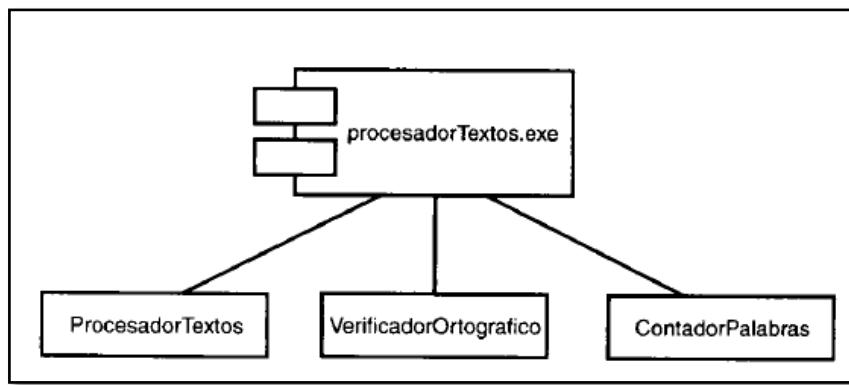


Figura N° 70: Adición de información al símbolo del componente
 (Kendall & Kendall, 2011)



6.5. Capacidades de hardware

Figura N° 71: Símbolos de relaciones entre componentes y las clases
 (Kendall & Kendall, 2011)

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria. Los estereotipos permiten precisar la naturaleza del equipo:

- Dispositivos
- Procesadores
- Memoria

Los nodos se interconectan mediante soportes bidireccionales que pueden a su vez estereotiparse. Esta vista permite determinar las consecuencias de la distribución y la asignación de recursos. Las instancias de los nodos pueden contener instancias de ejecución, como instancias de componentes y objetos. El modelo puede mostrar dependencias entre las instancias y sus interfaces, y también modelar la migración de entidades entre nodos u otros contenedores.

Los diagramas de despliegue muestran la configuración en funcionamiento del sistema, incluyendo su hardware y su software. Para cada componente de un diagrama de despliegue se deben documentar las características técnicas requeridas, el tráfico de red esperado, el tiempo de respuesta requerido, etc.

- Identificar los elementos del hardware que formarán parte del sistema
- Identificar los componentes que serán parte de cada nodo
- Identificar las relaciones que existe entre cada uno de estos (Dependencia, Interfaz, Dependencias-Interfaz)

6.6. Diagramas de despliegue: Nodos

Un Diagrama de Despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos (Figura N° 78).

- **Nodo:** Un Nodo es un elemento de hardware o software. Esto se muestra con la forma de una caja en tres dimensiones, como a continuación.

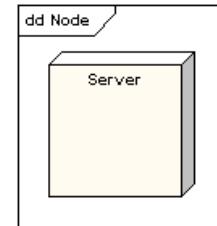


Figura N° 72: Nodo

- **Instancia de Nodo:** Una instancia de nodo se puede mostrar en un diagrama. Una instancia se puede distinguir desde un nodo por el hecho de que su nombre está subrayado y tiene dos puntos antes del tipo de nodo base. Una instancia puede o no tener un nombre antes de los dos puntos. El siguiente diagrama muestra una instancia nombrada de una computadora.

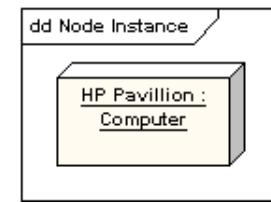


Figura N° 73: Instancia de Nodo

- **Estereotipo de Nodo:** Un número de estereotipos estándar se proveen para los nodos, nombrados «cdrom», «cd-rom», «computer», «disk array», «pc», «pc client», «pc server», «secure», «server», «storage», «unix server», «user pc». Estos mostraran un icono apropiado en la esquina derecha arriba del símbolo nodo.

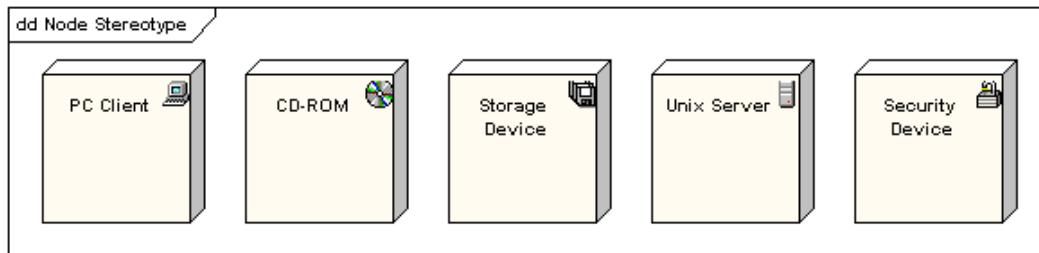


Figura N° 74: Estereotipo de Nodo

- **Artefacto:** Un artefacto es un producto del proceso de desarrollo de software, que puede incluir los modelos del proceso (modelos de Casos de Uso, modelos de Diseño, etc.), archivos fuente, ejecutables, documentos de diseño, reportes de prueba, prototipos, manuales de usuario y más. Un artefacto se denota por un rectángulo mostrando el nombre del artefacto, el estereotipo «artifact» y un icono de documento, como a continuación.

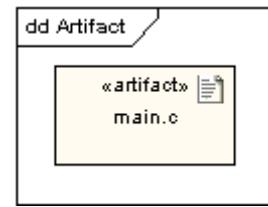


Figura N° 75: Artefacto

- **Asociación:** En el contexto del diagrama de despliegue, una asociación representa una ruta de comunicación entre los nodos. El siguiente diagrama muestra un diagrama de despliegue para una red, mostrando los protocolos de red como estereotipos y también mostrando multiplicidades en los extremos de la asociación.

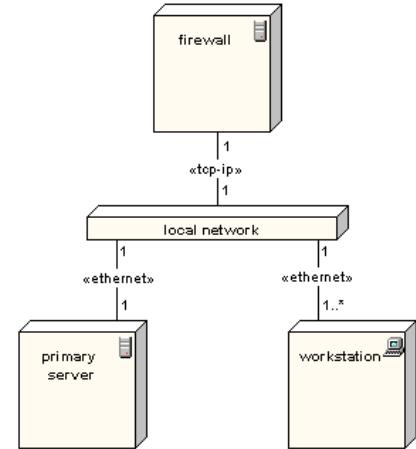


Figura N° 76: Asociación

- **Nodo como contenedor:** Un nodo puede contener otros elementos, como componentes o artefactos. El siguiente diagrama muestra un diagrama de despliegue para una parte del sistema embebido y muestra un artefacto ejecutable como contenido por el nodo madre (motherboard).

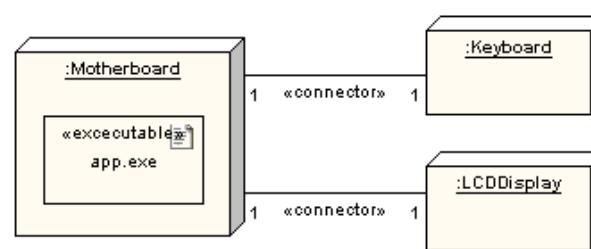


Figura N° 77: Nodo como Contenedor

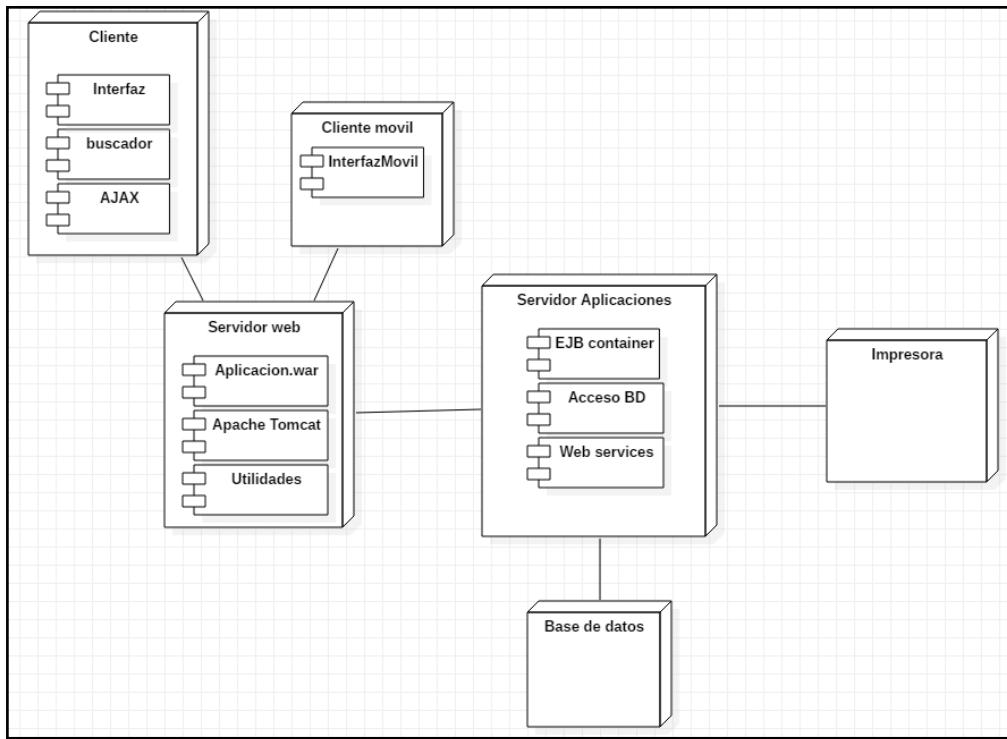


Figura N° 78: Ejemplo de diagrama de despliegue
 (Diagramas UML, s.f.)

6.7. Modelos complejos

Básicamente, un sistema puede definirse como un «conjunto de elementos en interacción» (Bertalanffy, 1968). Los sistemas complejos (p. ej. Organismos pluricelulares, colonias de hormigas, ecosistemas, economías, sociedades) están caracterizados por tener una estructura compuesta por varios niveles. En estos sistemas complejos:

- Los componentes de niveles jerárquicos inferiores suelen mostrar un grado de autonomía significativo.
- El comportamiento del sistema surge a partir de la autoorganización de sus componentes, sin que esta organización esté controlada ni dirigida por ningún ente exterior al sistema.
- Los componentes básicos de estos sistemas complejos (células, hormigas, individuos, poblaciones, empresa) perciben su entorno y responden a cambios en él de forma potencialmente diferente.

Por si esto fuera poco, muchos sistemas complejos son también adaptativos.

En estos sistemas adaptativos (organismos, ecosistemas, economías, sociedades...), el comportamiento de los componentes básicos del sistema puede evolucionar en el tiempo, dando lugar a una cierta capacidad de respuesta frente a cambios en el entorno por medio de mecanismos de:

- Aprendizaje a escala individual, y/o
- Selección y reemplazo (lo cual da lugar a un aprendizaje a escala poblacional)

Todas estas características hacen que el proceso de modelado formal de sistemas complejos difiera sustancialmente del de otros sistemas más simples. En particular, su naturaleza descentralizada, la presencia de bucles de causalidad y retroalimentación no lineales, y el hecho de contener varias unidades más o menos autónomas, que pueden interaccionar, evolucionar, y adaptar su comportamiento a cambios en el entorno, implica que en la mayoría de los casos es muy difícil —si no imposible— conseguir un modelo que pueda describir el sistema complejo adecuadamente y que además sea resoluble matemáticamente.

La posibilidad de trabajar con modelos formales más complejos que los modelos matemáticos tradicionales nos obliga a expandir ligeramente el marco del proceso de modelado científico que introdujimos en la sección anterior (ver Figura N° 79).

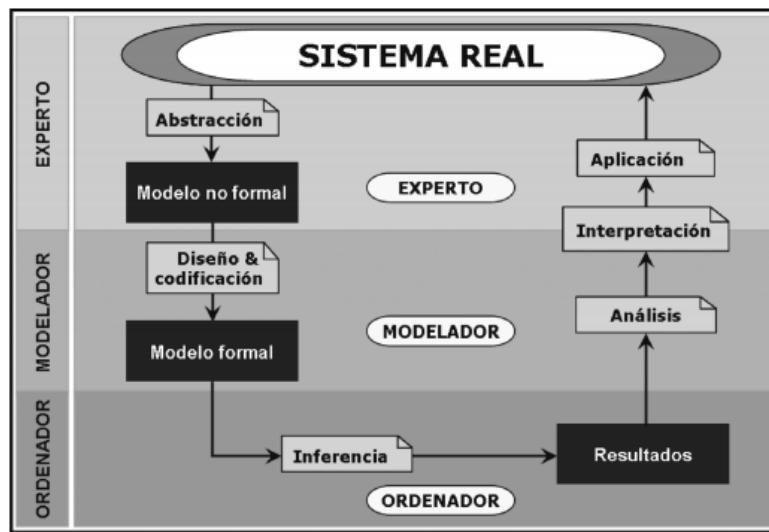


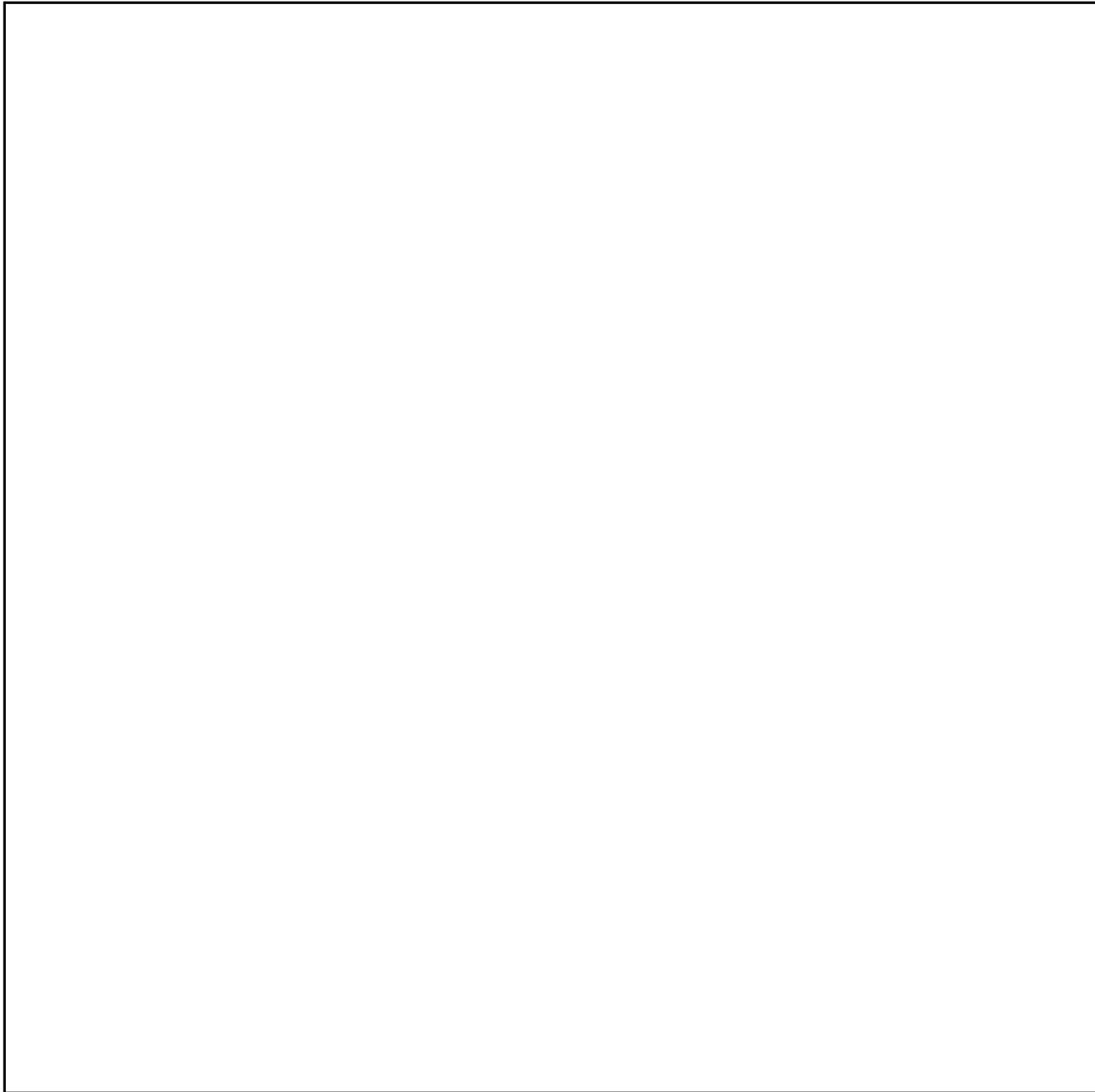
Figura N° 79: Proceso de modelado con abstracción intermedia
 (Empiria)

Actividad del Aprendizaje Esperado N° 6

Realiza el diagrama de colaboración para el proceso ver nota del siguiente ejemplo:

En nuestro instituto contamos con una plataforma de intranet para visualizar, el horario de clases y las notas correspondientes a cada módulo.

- El alumno Ingresa a la Intranet ingresando su nombre de usuario y su contraseña, una vez en la Intranet debe seleccionar el apartado que dice “Notas”, una vez ahí deberá seleccionar el Nombre del Módulo del cual desea visualizar sus notas.
- El MÓDULO mostrara las notas al alumno.



Aprendizaje Esperado 7

Realizan procesos de modelado, considerando herramientas CASE, modelos UML, capacidades de generación de código y de ingeniería inversa.

7.1 Herramientas de productividad CASE

El rápido incremento en las necesidades de software en las empresas causó que los desarrolladores de software empezaran a utilizar herramientas automatizadas como apoyo para minimizar la carga.

Hubo un gran auge en la creación de software cuando se empezó a generar con herramientas automatizadas, sin embargo, esto causó serios problemas pues existían millones y millones de líneas de código que necesitaban ser mantenidas y actualizadas. La industria de los computadores no podía cubrir el incremento de la demanda con los métodos que se estaban usando. Esto fue reconocido como una crisis de software. Para superar este problema en el proceso de desarrollo de software se introdujeron metodologías para crear estándares de desarrollo y se creó un soporte automatizado para el desarrollo y mantenimiento de software llamado Herramientas CASE.

Las herramientas CASE se definen como un conjunto de programas y procesos “guiados”, que ayudan a los analistas, desarrolladores, ingenieros de software y diseñadores en una o todas las etapas que comprende un ciclo de vida, con el objetivo de facilitar el desarrollo de software. El objetivo general de estas herramientas es acelerar el proceso para el que han sido diseñadas, es decir, para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Las herramientas CASE se diseñaron para aumentar la productividad en el desarrollo de software y reducir su costo.

7.1.1. Objetivos de las herramientas CASE

Automatizar:

- El desarrollo del software
- La documentación
- La generación del código
- En la búsqueda y corrección de errores
- La gestión del proyecto

Permitir:

- La reutilización del software
- La portabilidad del software
- La estandarización de la documentación

7.1.2. Clasificación de las Herramientas CASE

- Las plataformas que soportan
- Las fases del ciclo de vida del desarrollo de sistemas que cubren
- La arquitectura de las aplicaciones que producen
- Su funcionalidad
- Las fases del ciclo de vida del desarrollo de sistemas que cubren

El ciclo de vida de una aplicación o de un sistema de información se compone de varias etapas, que van desde la planificación hasta su implantación, mantenimiento y actualización. Los sistemas Case pueden cubrir la totalidad de estas fases o bien especializarse en algunas de ellas. En este último caso se pueden distinguir sistemas de "alto nivel" (Upper CASE), orientados a la autonomía y soporte de las actividades correspondientes a las dos primeras fases y, sistemas de "bajo nivel" (Lower CASE), dirigidos hacia las dos últimas. Los sistemas de "alto nivel" pueden soportar un número más o menos amplio de metodologías de desarrollo (Figura N° 80).

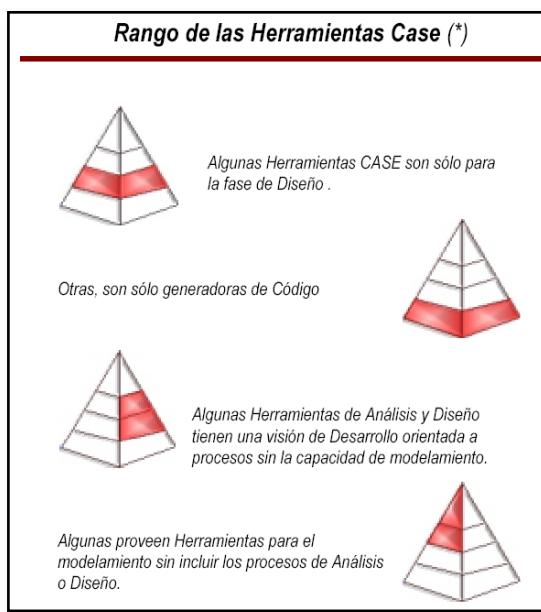


Figura N° 80: Herramientas CASE en base a las fases que cubren
 (INEI)

Se distinguen tres grupos de herramientas CASE en base a las fases del ciclo de vida que cubren en el desarrollo del sistema, y son los siguientes:

- **CASE de alto nivel (Upper CASE):** Son aquellas herramientas que automatizan o apoyan las fases finales o superiores del ciclo de vida del desarrollo de sistemas como la planificación de sistemas, el análisis de sistemas y el diseño de sistemas.

- **CASE de bajo nivel (Lower CASE):** Son aquellas herramientas que automatizan o apoyan las fases finales o inferiores del ciclo de vida como el diseño detallado de sistemas, la implantación de sistemas y el soporte de sistemas.
- **Herramientas CASE Integradas (Integrated Case):** Abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.

7.1.3. Ventajas y Desventajas de Herramientas CASE

Tipo De Case	Ventajas	Desventajas
Integrated - Case	<ul style="list-style-type: none"> • <i>Integra el ciclo de vida</i> • <i>Permite lograr importantes mejoras de productividad a mediano plazo</i> • <i>Permite un eficiente soporte al mantenimiento de sistemas</i> • <i>Mantiene la consistencia de los sistemas a nivel corporativo</i> 	<ul style="list-style-type: none"> • <i>No es tan eficiente para soluciones simples, sí no para soluciones complejas</i> • <i>Depende del hardware y software es costoso</i>
Upper Case	<ul style="list-style-type: none"> • <i>Se utiliza en plataforma PC, es aplicable a diferentes entornos</i> • <i>Menor costo</i> 	<ul style="list-style-type: none"> • <i>Permite mejorar la calidad de los sistemas, pero no la productividad</i> • <i>No permite la integración de ciclo de vida</i>
Lower Case	<ul style="list-style-type: none"> • <i>Permite lograr importantes mejoras de productividad a corto plazo</i> • <i>Permite un eficiente soporte el mantenimiento de sistemas</i> 	<ul style="list-style-type: none"> • <i>No garantiza la consistencia de los resultados a nivel corporativo</i> • <i>No garantiza la eficiencia de análisis y diseño</i> • <i>No permite la integración de ciclo de vida</i>

7.1.4. Herramientas Case más utilizadas

<ul style="list-style-type: none"> - ER win - ArgoUML - Easy Case - Oracle Designer - Power Designer - System Architect - SNAP - Microsoft visio 	<ul style="list-style-type: none"> - NetBeans - Eclipse - OmniGraffle - Serena Composer - GUI Design Studio - Eclipse Indigo *Plug-in "UML 2 Tools" - Expression Web 4 	<ul style="list-style-type: none"> - Edraw - Mockflow - yUML - Oracle SQL Developer - CASE Studio 2 - Poseidon - Sharepoint workflow - SQL server
--	---	---

7.2. Funciones de modelado y reglas UML

La finalidad de UML según OMG

El OMG define los propósitos de UML de la siguiente manera:

Brindar a arquitectos de sistemas, ingenieros y desarrolladores de software las herramientas para el análisis, el diseño y la implementación de sistemas basados en software, así como para el modelado de procesos de negocios y similares.

Hacer progresar el estado de la industria permitiendo la interoperabilidad de herramientas de modelado visual de objetos. No obstante, para habilitar un intercambio significativo de información de modelos entre herramientas, se requiere de un acuerdo respecto de la semántica y notación.

UML cumple con los siguientes requerimientos:

Establecer una definición formal de un metamodelo común basado en el estándar MOF (Meta-Object Facility) que especifique la sintaxis abstracta del UML. La sintaxis abstracta define el conjunto de conceptos de modelado UML, sus atributos y sus relaciones, así como las reglas de combinación de estos conceptos para construir modelos UML parciales o completos.

Especificar los elementos de notación de lectura humana para representar los conceptos individuales de modelado UML, así como las reglas para combinarlos en una variedad de diferentes tipos de diagramas que corresponden a diferentes aspectos de los sistemas modelados.

Definir formas que permitan hacer que las herramientas UML cumplan con esta especificación. Esto se apoya (en una especificación independiente) con una especificación basada en XML de formatos de intercambio de modelos correspondientes (XMI) que deben ser concretados por herramientas compatibles.

Reglas de UML

- Los bloques de construcción de UML no pueden simplemente combinarse de cualquier manera.
- Como cualquier lenguaje, UML tiene un número de reglas que especifican a qué debe parecerse un modelo bien formado.
- Un Modelo bien formado es aquel que es semánticamente autoconsistente y está en armonía con todos sus modelos relacionados.
- Los modelos que no llegan a ser bien formados son inevitables, conforme los detalles de un sistema van a apareciendo y mezclándose durante el proceso de desarrollo de software.
- Las reglas de UML estimulan (pero no obligan) a considerar las cuestiones más importantes de análisis, diseño e implementación que llevan a tales sistemas a convertirse en bien formados con el paso del tiempo.

Son reglas sintácticas y semánticas:

- Nombres: Cómo llamar a los elementos, relaciones y diagramas.
- Alcance: El contexto que da un significado específico a un nombre.
- Visibilidad: Cómo se pueden ver y utilizar esos nombres por otros.
- Integridad: Cómo se relacionan apropiada y consistentemente unos elementos con otros.
- Ejecución: Qué significa ejecutar o simular un modelo dinámico.

Los modelos construidos durante el desarrollo de un sistema con gran cantidad de software tienden a evolucionar, por lo que además de modelos bien formados hay otros que pueden ser:

- Abreviados
- Incompletos
- Inconsistentes

Su solución es considerar las cuestiones más importantes de análisis, diseño e implementación.

7.3. Generadores de código

Uno de los temas que más de moda está en los últimos años, son los generadores de código de manera automática, que toman como entrada una descripción del diseño que se quiere construir y ellos realizan una implementación en el lenguaje que nosotros le indiquemos.

En un primer momento, los generadores de código nacen como una manera de agilizar el desarrollo, por ejemplo, si siempre partimos de un caso en el que usamos cinco librerías y nuestro diseño es de cierta manera, para nosotros sería muy útil, poder disponer de esa plantilla ya creada y luego completar el contenido. Este proceso es muy importante si queremos diseñar un prototipo rápido, de lo que va a ser nuestro servicio y con el cual le podemos enseñar a un posible cliente, inversor o al propio equipo, cómo será lo que queremos montar.

7.3.1. Tipos de generadores de código

Dentro de los generadores de código, podemos distinguir dos tipos principales:

- **Generadores interactivos:** Este tipo de generadores es muy común actualmente y permite que, con un simple sistema de arrastrar y configurar un par de parámetros del elemento, se pueda generar todo el código necesario para implementar esa funcionalidad.

Un ejemplo de ello es el App Studio (Figura N° 81), el generador de Aplicaciones para Windows Phone que Microsoft pone a disposición de cualquier persona.

- **Generadores usando un lenguaje de modelado:** Este tipo de generadores son menos comunes, pero son los más “potentes”, ya que usando una descripción del modelo que queremos crear en un lenguaje de modelado como UML, son capaces de crear un porcentaje bastante amplio del código.

Un ejemplo de esto es Visual Paradigm, y una de sus ventajas es que permite generar código en diversos lenguajes como C++, Java y PHP.

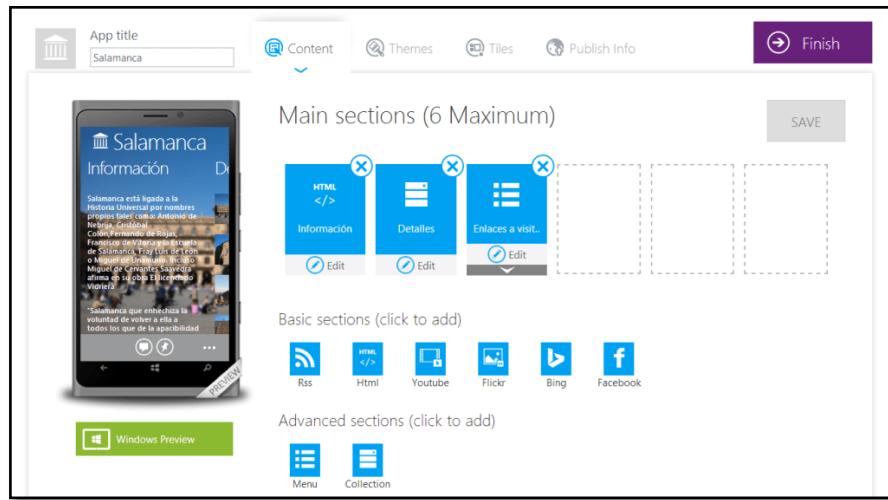


Figura N° 81: App Studio
 (Durán, 2014)

7.4. Ingeniería inversa

La ingeniería inversa (en inglés, reverse engineering) es el proceso o técnica de descubrir los principios tecnológicos de un producto, herramienta, dispositivo o sistema. Este descubrimiento se puede realizar mediante el razonamiento abductivo (haciendo conjeturas) de su estructura, función y operación.

Los productos más típicamente sometidos a esta técnica son los programas para computador, pero cualquier producto puede ser objeto de análisis de ingeniería inversa. Se denomina así porque avanza en dirección opuesta a las tareas habituales de ingeniería, que consisten en utilizar datos técnicos para elaborar un producto determinado.

Aplicar ingeniería inversa a algo supone profundizar en el estudio de su funcionamiento, hasta el punto de que podamos llegar a entender, modificar y mejorar dicho modo de funcionamiento. La aplicación de ingeniería inversa nunca cambia la funcionalidad del producto objeto de la aplicación, sino que permite obtener productos que indican cómo se ha construido el mismo.

Su realización permite obtener los siguientes beneficios:

- Reducir la complejidad del sistema
- Generar diferentes alternativas
- Recuperar y/o actualizar la información perdida (cambios que no se documentaron en su momento)
- Detectar efectos laterales
- Facilitar la reutilización

Existen los siguientes tipos de ingeniería inversa:

- Datos
- Lógica o proceso
- Interfaces de usuario

7.4.1. Herramientas para ingeniería inversa

Cuando un desarrollador crea un programa, comienza escribiendo el código fuente en el lenguaje de programación seleccionado para acabar compilándolo y crear un programa ejecutable. Llegado este punto, nadie podría editar el programa sin disponer del código fuente inicial para realizar una nueva compilación de este, y en esto es precisamente en lo que se basa el cracking.

La técnica de ingeniería inversa constituye la piedra angular del cracking. Se basa en la descompilación, o compilación inversa, de un programa a un lenguaje de programación, generalmente, el más básico que existe que es el lenguaje ensamblador.

A continuación, veremos las principales herramientas utilizadas para este proceso:

7.4.2. Depuradores

Un depurador es un programa utilizado para controlar a otros programas. Permite analizar un código paso a paso y establecer puntos de control para buscar posibles fallos.

Lo más interesante de los depuradores son los puntos de ruptura (breakpoint) ya que permiten detener la ejecución de un programa cuando se cumpla una condición y ver el estado de todas las variables y la memoria en dicho momento.

Los principales depuradores para Windows son:

- **OllyDbg:** Depurador de código ensamblador de 32 bits para sistemas operativos Microsoft Windows. Pone especial énfasis en el análisis del código binario, esto lo hace muy útil cuando no está disponible el código fuente del programa.
- **WinDBG:** Es una pieza de software gratuita de Microsoft que puede ser usada para depuración local en modo usuario, o incluso depuración remota en modo Kernel.
- **Visual DuxDebugger:** Desensamblador depurador de 64 bits para Windows, especialmente útil cuando el código fuente no está disponible.
- **GNU Debugger (gdb):** Es un depurador portable que se puede utilizar en varias plataformas Unix y funciona para varios lenguajes de programación como C, C++.
- **Fortran (gdb):** Ofrece la posibilidad de trazar y modificar la ejecución de un programa. El usuario puede controlar y alterar los valores de las variables internas del programa.

- **SoftICE:** Es un depurador en modo kernel propietario y de pago para Microsoft Windows. SoftICE es capaz de suspender todas las operaciones en Windows cuando se deseé, lo cual resulta útil para depurar drivers, ya que es importante conocer cómo se accede al hardware, así como las funciones del sistema operativo.

7.4.3. Desensambladores

Un desensamblador es exactamente lo contrario de un ensamblador. Tal como un ensamblador convierte código escrito en ensamblador en código máquina binario, un desensamblador invierte el proceso e intenta recrear el código en ensamblador partiendo del código máquina binario.

Los desensambladores intentan, por tanto, mostrar el código del lenguaje de la máquina en un formato más amigable a los ojos de los humanos. A continuación, indicaremos los principales desensambladores según el sistema operativo:

Windows	Linux
IDA Pro PE Explorer W32DASM IDA 6.6 IDA Pro Freeware 5.0 BORG disassembler HT Editor diStorm64	Bastard Disassembler Ciasdis Objdump GDB LDasm Mac OS GDB Machonist Otool Ndisasm

7.4.4. Descompiladores o compiladores inversos

Un descompilador es un programa de ordenador que realiza la operación inversa a un compilador, es decir, traduce código de bajo nivel de abstracción a un lenguaje de mayor nivel de abstracción.

Por lo tanto, un descompilador toma el código binario ejecutable e intenta recrear el código fuente de alto nivel a partir de él.

A continuación, vemos algunos de los descompiladores más utilizados:

- **DCC Decompiler:** Solo permite descompilar ejecutables DOS basados en el 80286 y lenguaje C. El enfoque usado por DCC en la descompilación se basa en la teoría y técnicas de optimización de grafos adoptadas por los compiladores.

- **Boomerang Decompiler Project:** Es programa de código abierto para descompilar programas en C. Altera la semántica de cada instrucción en ensamblador e implementa el análisis estático de flujo de datos.
- **Reverse Engineering Compiler (REC):** Descompila código ensamblador a una representación del código semejante a C, que requiere la edición manual para volver a compilar. El código está a medio camino entre ensamblador y C, pero es mucho más legible que el ensamblador puro.

7.5. Código fuente

7.5.1. ¿Qué es el código fuente?

Los computadores, ya sean PC domésticos, modernos móviles u computadores para la industria o la ciencia, trabajan con un sistema binario: Encendido/apagado, cargado/no cargado, 1/0. Una secuencia de estados (bits) indica al computador lo que tiene que hacer. Mientras que en los comienzos de la tecnología informática se creaban comandos con estas dos condiciones, hace tiempo que se ha pasado a escribir aplicaciones en un lenguaje de programación legible por los humanos. En este contexto, “legible por los humanos” es antónimo de “legible por las máquinas”. Mientras que los computadores solo trabajan con valores numéricos, las personas utilizamos palabras para comunicarnos. Al igual que las lenguas extranjeras, los lenguajes de programación también tienen que aprenderse antes de utilizarse.

Hay cientos de lenguajes de programación diferentes y no se puede decir que unos sean mejores que otros, pues esto depende del contexto del proyecto y de la aplicación para la que se use el código fuente. Entre los lenguajes de programación más conocidos se encuentran:

- BASIC
- Java
- C
- C++
- Pascal
- Python
- PHP
- JavaScript

Para que los computadores puedan comprenderlos, estos deben traducirse al código de máquina. Para que los computadores puedan procesar el texto fuente creado por los programadores tiene que haber un traductor entre ambos en forma de programa adicional. Esta aplicación auxiliar puede presentarse como compilador o como intérprete:

- **Compilador:** Este tipo de aplicación traduce (compila) el código fuente en un código que el procesador puede comprender y ejecutar. Este código de máquina se almacena en forma de archivo ejecutable.

- **Intérprete:** Un intérprete traduce el código fuente línea a línea y lo ejecuta directamente. El proceso de traducción funciona mucho más rápido que en un compilador, pero la ejecución es más lenta y se necesita una gran cantidad de memoria.

Con todo, la elección de uno u otro no es libre, pues es el lenguaje de programación el que determina si debe utilizarse un compilador o un intérprete, aunque hoy en día cada vez es más frecuente recurrir a una solución provisional: Just-in-time (JIT) compilation, en español compilación en tiempo de ejecución. Este tipo de traducción intenta combinar las ventajas de ambos programas (análisis y ejecución rápidos) y se emplea, por ejemplo, en los navegadores para gestionar JavaScript, PHP o Java más eficazmente.

7.5.2. Estructura del texto fuente

Al escribir programas hay que cumplir determinadas convenciones independientemente del lenguaje de programación que se emplee. Muy pocos lenguajes se crean de la nada, sino que la mayoría se desarrolla a partir de los otros, de ahí que haya determinados elementos que aparecen reiteradamente en diferentes códigos de programación:

- **Comandos:** Las instrucciones son probablemente la base de todas las aplicaciones. Con ellas, los programadores indican a sus futuros programas qué es lo que tienen que hacer. Tales comandos pueden, por ejemplo, desencadenar determinados procesos de cálculo o incluso mostrar un texto.
- **Variables:** Las variables son espacios en los que se insertan datos. Dentro del código fuente se hace referencia a estas una y otra vez mediante la asignación de un nombre.
- **Comparaciones:** Especialmente decisivas para la estructura de la mayoría de programas son las consultas que funcionan según un esquema causa-efecto, es decir, siguiendo el principio de la lógica proposicional. La introducción de un valor lógico determinado desencadena un evento; si no, se produce uno diferente.
- **Bucles:** Las consultas también pueden constituir la base para los bucles del texto fuente. Un comando se repite hasta que se haya alcanzado un valor determinado, tras lo cual el programa abandonará el bucle y ejecutará el resto del código.
- **Comentarios:** Todos los lenguajes de programación actuales permiten comentar líneas dentro del código, con lo que es posible escribir texto en el código fuente que el programa no tiene en cuenta. El autor puede así introducir comentarios en el texto fuente para que él mismo u otro desarrollador puedan entender en el futuro las diferentes partes del código.

La creación de código fuente siempre está ligada a un problema. Los desarrolladores escriben programas para ofrecer soluciones, pero no se establece cuál es el camino para ello. Cuando dos programadores se ocupan del mismo problema, puede ocurrir que ambos textos fuente difieran significativamente entre sí, incluso a pesar de que se trabaje con el mismo lenguaje.

A pesar de que en muchos casos no hay una solución única, todas las tareas de programación tienen algo en común: Un buen texto fuente debe prescindir de código innecesario, pues este hace que el programa sea más complejo, lento y propenso a errores. El código fuente poco claro que ni siquiera los profesionales pueden comprender se denomina código espagueti, pues su estructura es tan confusa como un montón de espaguetis en un plato.

```
public int SerieFibonacci (int limiteSerie) {  
    int result=-1, a, b;  
    if(limiteSerie==0 || limiteSerie==1) {  
        result=limiteSerie;  
    }  
    else {  
        a = -1;  
        b = 1;  
        for(int i = 0; i <= limiteSerie; i++) {  
            result = a + b;  
            a = b;  
            b = result;  
        }  
    }  
    return result;  
}
```

Figura N° 82: Código Fuente
(Digital Guide IONOS, 2020)

7.6. Obtención de modelos de software

Un modelo para el desarrollo de software es una representación abstracta de un proceso. Cada modelo representa un proceso desde una perspectiva particular y así proporcione información parcial sobre el proceso. Estos modelos generales no son descripciones definitivas de los procesos del software, más bien son abstracciones de los procesos que se pueden utilizar para el desarrollo del software. Puede pensarse en ellos como marcos de trabajo del proceso y que pueden ser adaptados para crear procesos más específicos. Los modelos que mencionaremos en este punto son:

- 1) **El modelo en cascada.** Considera las actividades fundamentales del proceso especificación, desarrollo, validación y evolución. Los representa como fases separadas del proceso, tales como la especificación de requerimientos, el diseño del software, la implementación, las pruebas, etc.
- 2) **El modelo de desarrollo evolutivo (espiral).** Este enfoque entrelaza las actividades especificación, desarrollo y validación. Es decir, surge de un sistema inicial que se desarrolla

rápidamente a partir de especificaciones abstractas. Basándose en las peticiones del cliente para producir un sistema que satisfaga sus necesidades.

3) El modelo de desarrollo basado en componentes. Este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo se enfoca en integrar estos componentes en el sistema más que en desarrollarlos desde cero. Estos tres modelos se utilizan ampliamente en la práctica actual de la ingeniería del software, no se excluyen mutuamente y a menudo se utilizan juntos especialmente para el desarrollo de grandes sistemas.

Actividad del Aprendizaje esperado N° 7

Utilizando cualquiera de las herramientas Case, vistas en apartado anterior, intenta crear el diagrama de clases del siguiente enunciado:

Diseñar una aplicación orientada a objetos que describa la siguiente situación: Una tienda de Retail tiene a la venta distintos productos, en sus departamentos de Tecnología, Computación y electrodomésticos, cada producto tiene características y funciones que lo diferencian de otro y también se puede clasificar por departamento corresponde.

Los Artículos mas vendidos son los Siguientes: Computador (Notebook, Desktop), Impresora, Celular, Lavadora, Televisor, Tablet, Reloj inteligente.

Especificar la jerarquía de herencia, las clases, los atributos y los métodos de cada clase.

III UNIDAD DE APRENDIZAJE: INTRODUCCIÓN A ARQUITECTURA DE SISTEMAS

Aprendizaje Esperado 8

Analizan fundamentos de arquitectura de software, considerando objetivos, perspectivas de diseño y ventajas de patrones de diseño.

8.1. Objetivos de diseño de arquitectura de software

El diseño de la arquitectura de software¹⁸ se realiza por lo habitual siguiendo un enfoque de “divide y vencerás”. El problema general, que es realizar el diseño de toda la arquitectura, se divide en problemas de menor tamaño, que son realizar el diseño de partes de la arquitectura, y que pueden ser resueltos de manera más fácil.

Este procedimiento consiste en elegir en cada iteración un subconjunto de todos los drivers de la arquitectura y tomar decisiones de diseño al respecto, lo cual resulta en estructuras que permiten satisfacerlos. El diseño resultante se evalúa, se elige enseguida otro subconjunto de los drivers, y se procede de la misma manera hasta que se completa el diseño. El proceso termina cuando se considera que se han tomado suficientes decisiones de diseño para satisfacer el conjunto de drivers, o bien, cuando concluye el tiempo que el arquitecto tiene asignado para realizar las actividades de diseño.

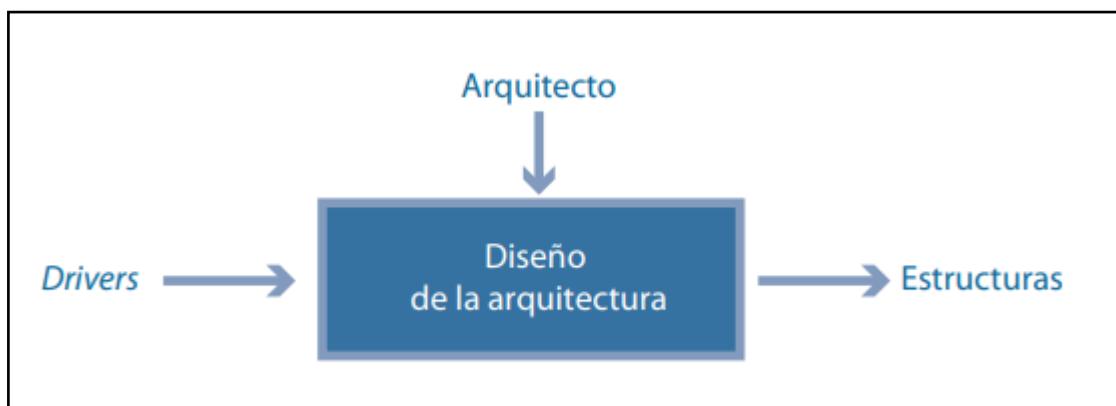


Figura N° 83: Diseño como caja negra (las entradas están a la izquierda, las salidas, a la derecha, y los responsables, arriba)
(Propia, Elaboración, s.f.)

¹⁸ Diseño Arquitectónico de Software:

<http://upiicsa.tecnologia-educativa.com.mx/docs/u2/s3/DISENO%20ARQUITECTONICO.pdf>

En el diseño de la arquitectura de software, una de las ventajas de seguir el enfoque “divide y vencerás”, es que resulta más simple y realista diseñar de forma iterativa e incremental, que tratar de tomar todos los drivers y, de una sola vez, producir un diseño que los satisfaga a todos.

La segunda iteración de diseño se enfoca en un subconjunto de los casos de uso, mientras que la tercera lo hace en un atributo de calidad: El desempeño.

El proceso de diseño involucra la toma de decisiones. Dado que los sistemas rara vez son completamente innovadores, muchos de los problemas de diseño con los que se enfrenta el arquitecto ya han sido atacados previamente por otras personas. Por ello, una parte considerable de la toma de decisiones involucra la identificación, selección y adecuación de soluciones existentes con el fin de resolver los subproblemas de diseño que se deben atender como resultado de seguir el enfoque “divide y vencerás”.

Una vez que se eligen soluciones existentes y se adecúan, se producen estructuras enfocadas al problema que se está resolviendo y las cuales son parte del conjunto de estructuras lógicas, dinámicas o físicas de la arquitectura del sistema.

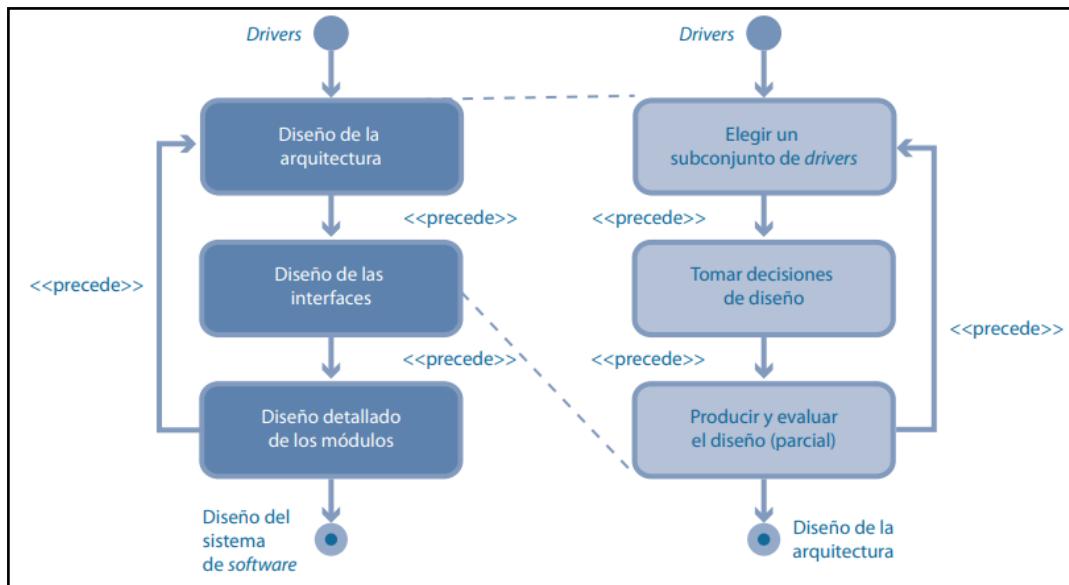


Figura N° 84: Proceso general de diseño de la arquitectura.
 (Propia, Elaboración, s.f.)

8.2. Fundamentos de la arquitectura de software

No podemos hablar de arquitectura del software sin relacionarlo con ingeniería de software, ya que el diseño de software se encuentra técnicamente relacionado con Ingeniería de sistemas cuyo objetivo es producir software de alta calidad viendo al diseño como el “plano del software” destacando las áreas de aplicación: Datos, arquitectura, interfaces y los componentes.

El diseño de datos transforma el dominio de información en estructuras de datos, el diseño arquitectónico define la relación entre elementos estructurales del software. Esto se conoce como “diseño de interfaz” es decir como el software se comunicará dentro de sí mismo, con otros sistemas y con las personas.

Principios básicos para lograr un buen diseño.

- **Abstracción:** Descripción simplificada o especificación que enfatiza algunos de los detalles o propiedades del sistema, mientras suprime otros. La abstracción puede ser de datos, procedimientos y control.
- **Refinamiento:** Es la base del diseño. Es un proceso de elaboración que comienza con un nivel de abstracción alto y va descendiendo sucesivamente de nivel de abstracción hasta llegar a un nivel bajo. El refinamiento tiene como objetivo encontrar detalles de bajo nivel que podrían ser difíciles de plasmar.
- **Modularidad:** Se trata de dividir el software en componentes nombrados y abordados por separado llamados “módulos”, que se integran para resolver los requisitos del problema. Aunque la modularidad sea benéfica para un sistema, tampoco debe exagerarse en el número de módulos, lo ideal es diseñar pocas interfaces, estrecha comunicación, lograr interfaces explícitas.
- **Ocultamiento de información:** Los módulos de un sistema deben diseñarse de modo que la información contenida en ellos sea inaccesible a todos aquellos módulos que no necesiten tal información.
- **Visibilidad** es el conjunto de componentes de un programa que pueden ser invocados o utilizados para el uso de sus datos por un componente de manera directa.
- **Cohesión:** Es una extensión del principio de ocultación de información. Esta se obtiene cuando el contenido de un módulo está diseñado para realizar una tarea sencilla sin depender de otros módulos; o bien que un módulo lleve a cabo solo una función de procesamiento.
- **El acoplamiento** se refiere a la fuerza de la relación entre módulos de un sistema. En general, los buenos diseñadores buscan desarrollar la estructura de un sistema de tal forma que un módulo tenga poca dependencia de cualquier otro módulo. Es necesario tener un bajo acoplamiento.

8.3. Decisiones en el diseño de arquitectura de software

Durante el proceso de diseño arquitectónico, los arquitectos del sistema deben tomar algunas decisiones estructurales que afectarán profundamente el sistema y su proceso de desarrollo. Con base en su conocimiento y experiencia, deben considerar las siguientes preguntas fundamentales sobre el sistema:

1. *¿Existe alguna arquitectura de aplicación genérica que actúe como plantilla para el sistema que se está diseñando?*
2. *¿Cómo se distribuirá el sistema a través de algunos núcleos o procesadores?*
3. *¿Qué patrones o estilos arquitectónicos pueden usarse?*
4. *¿Cuál será el enfoque fundamental usado para estructurar el sistema?*
5. *¿Cómo los componentes estructurales en el sistema se separarán en subcomponentes?*
6. *¿Qué estrategia se usará para controlar la operación de los componentes en el sistema?*
7. *¿Cuál organización arquitectónica es mejor para entregar los requerimientos no funcionales del sistema?*
8. *¿Cómo se evaluará el diseño arquitectónico?*
9. *¿Cómo se documentará la arquitectura del sistema?*

La arquitectura de un sistema de software puede basarse en un patrón o un estilo arquitectónico particular. Un patrón arquitectónico es una descripción de una organización del sistema, tal como una organización cliente-servidor o una arquitectura por capas. Los patrones arquitectónicos captan la esencia de una arquitectura que se usó en diferentes sistemas de software.

Es necesario elegir la estructura más adecuada, como cliente-servidor o estructura en capas, que le permita satisfacer los requerimientos del sistema. Para descomponer las unidades del sistema estructural, usted opta por la estrategia de separar los componentes en subcomponentes.

Debido a la estrecha relación entre los requerimientos no funcionales y la arquitectura de software, el estilo y la estructura arquitectónicos particulares que se elijan para un sistema dependerán de los requerimientos de sistema no funcionales:

- 1. Rendimiento:** Si el rendimiento es un requerimiento crítico, la arquitectura debe diseñarse para localizar operaciones críticas dentro de un pequeño número de componentes, con todos estos componentes desplegados en la misma computadora en vez de distribuirlos por la red.
- 2. Seguridad:** Si la seguridad es un requerimiento crítico, será necesario usar una estructura en capas para la arquitectura, con los activos más críticos protegidos en las capas más internas, y con un alto nivel de validación de seguridad aplicado a dichas capas.
- 3. Protección:** Si la protección es un requerimiento crítico, la arquitectura debe diseñarse de modo que las operaciones relacionadas con la protección se ubiquen en algún componente individual o en un pequeño número de componentes. Esto reduce los costos y problemas de validación de la

protección, y hace posible ofrecer sistemas de protección relacionados que, en caso de falla, desactiven con seguridad el sistema.

4. Disponibilidad: Si la disponibilidad es un requerimiento crítico, la arquitectura tiene que diseñarse para incluir componentes redundantes de manera que sea posible sustituir y actualizar componentes sin detener el sistema.

5. Mantenibilidad: Si la mantenibilidad es un requerimiento crítico, la arquitectura del sistema debe diseñarse usando componentes autocontenidos de grano fino que puedan cambiarse con facilidad.

Evaluar un diseño arquitectónico es difícil porque la verdadera prueba de una arquitectura es qué tan bien el sistema cubre sus requerimientos funcionales y no funcionales cuando está en uso. Sin embargo, es posible hacer cierta evaluación al comparar el diseño contra arquitecturas de referencia o patrones arquitectónicos genéricos.

8.4. Perspectivas de arquitectura de software

- Una arquitectura de software es una descripción de cómo se organiza un sistema de software. Las propiedades de un sistema, como rendimiento, seguridad y disponibilidad, están influidas por la arquitectura utilizada.
- Las decisiones de diseño arquitectónico incluyen decisiones sobre el tipo de aplicación, la distribución del sistema, los estilos arquitectónicos a usar y las formas en que la arquitectura debe documentarse y evaluarse.
- Las arquitecturas pueden documentarse desde varias perspectivas o diferentes vistas. Las posibles vistas incluyen la conceptual, la lógica, la de proceso, la de desarrollo y la física.
- Los patrones arquitectónicos son medios para reutilizar el conocimiento sobre las arquitecturas de sistemas genéricos. Describen la arquitectura, explican cuándo debe usarse, y exponen sus ventajas y desventajas.
- Los patrones arquitectónicos usados comúnmente incluyen el modelo de vista del controlador, arquitectura en capas, repositorio, cliente-servidor, y tubería y filtro.
- Los modelos genéricos de las arquitecturas de sistemas de aplicación ayudan a entender la operación de las aplicaciones, comparar aplicaciones del mismo tipo, validar diseños del sistema de aplicación y valorar componentes para reutilización a gran escala.
- Los sistemas de procesamiento de transacción son sistemas interactivos que permiten el acceso y la modificación remota de la información, en una base de datos por parte de varios usuarios.

- Los sistemas de información y los sistemas de gestión de recursos son ejemplos de sistemas de procesamiento de transacciones.
- Los sistemas de procesamiento de lenguaje se usan para traducir textos de un lenguaje a otro y para realizar las instrucciones especificadas en el lenguaje de entrada. Incluyen un traductor y una máquina abstracta que ejecuta el lenguaje generado.

8.5. Principio de los patrones en arquitectura de software

Los patrones arquitectónicos se abocan a un problema de aplicación específica dentro de un contexto dado y sujeto a limitaciones y restricciones. El patrón propone una solución arquitectónica que sirve como base para el diseño de la arquitectura.

Por ejemplo, el estilo de arquitectura general para una aplicación podría ser el de llamar y regresar o el que está orientado a objeto. Pero dentro de ese estilo se encontrará un conjunto de problemas comunes que se abordan mejor con patrones arquitectónicos específicos.

Los patrones de arquitectura son formatos preparados que resuelven problemas arquitectónicos recurrentes. Una infraestructura de la arquitectura (middleware) es un conjunto de componentes sobre los que puede construir una cierta clase de arquitectura. Muchas de las principales dificultades arquitectónicas deberían resolverse en la infraestructura, normalmente destinada a un dominio específico: mandato y control, MIS, sistema de control, etc.

La tabla muestra las categorías que se presentan y los patrones que contienen.

Categoría	Patrón
Estructura	Capas Conductos y filtros Pizarra
Sistemas distribuidos	Intermediario
Sistemas interactivos	Modelo-Vista-Controlador Presentación-Abstracción-Control
Sistemas adaptables	Reflejo Microkernel

Los patrones se presentan con el siguiente formato, muy utilizado:

- Nombre del patrón
- Contexto
- Problema
- Fuerza la descripción de diferentes aspectos del problema que se deberían tener en cuenta
- Solución
 - Fundamentos
 - Contexto resultante

Actividad del aprendizaje Esperado N° 8

Responda las siguientes preguntas:

1. ¿Qué es la Arquitectura de Software?

- A) Es la manera en la que un componente no se relaciona en una estructura
- B) Es la manera en la que los distintos componentes se relacionan para crear una estructura
- C) Es la manera en la que un componente se relaciona con uno similar
- D) Ninguna de las anteriores

2. ¿Qué es una descripción arquitectónica?

- A) Es un conjunto de productos externos de un trabajo terminado que no reflejan puntos de vistas de un sistema
- B) Son partes separadas de un producto terminado que no dan puntos de vista y se elaboran independientemente
- C) Es un conjunto de productos de trabajo que reflejan puntos de vista distintos del sistema
- D) Ninguna de las anteriores

3. ¿Qué permite la representación de la Arquitectura de Software?

- A) Representan el desarrollo de estructuras multinacionales
- B) Crean estructuras para la elaboración de un sistema de software
- C) Permiten la comunicación entre todas las partes
- D) Permiten la comunicación entre una parte solamente

4. Es el nivel en el que el arquitecto usa un diagrama de contexto

- A) Nivel de diseño Arquitectónico
- B) Nivel de Estructura
- C) Nivel de Liderazgo

Aprendizaje Esperado 9

Aplican distintos tipos de patrones en modelado de arquitectura de software.

9.1. Patrón Modelo-Vista-Controlador (MVC)¹⁹

Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Es un modelo que ha demostrado su validez en todo tipo de aplicaciones, lenguajes y plataformas de desarrollo.

9.1.1. El Modelo

El Modelo representa los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. Es el responsable de acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.

- El Modelo define la funcionalidad del sistema (reglas de negocio). Por ejemplo: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
- El Modelo lleva un registro de las vistas y controladores del sistema, si es un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo, por ejemplo: un fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, etc.

9.1.2. El Controlador

El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El controlador es responsable de Recibir los eventos de entrada (un clic, un cambio en un campo de texto, etc.). Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar ()". Una petición al modelo puede ser "Obtener_tiempo_de_entrega (nueva_orden_de_venta)".

¹⁹Patrón Vista Controlador: <https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>

9.1.4. Las Vistas

La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con este.

Las vistas son responsables de Recibir datos del modelo y los muestra al usuario. Tienen un registro de su controlador asociado (normalmente porque además lo instancia). Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

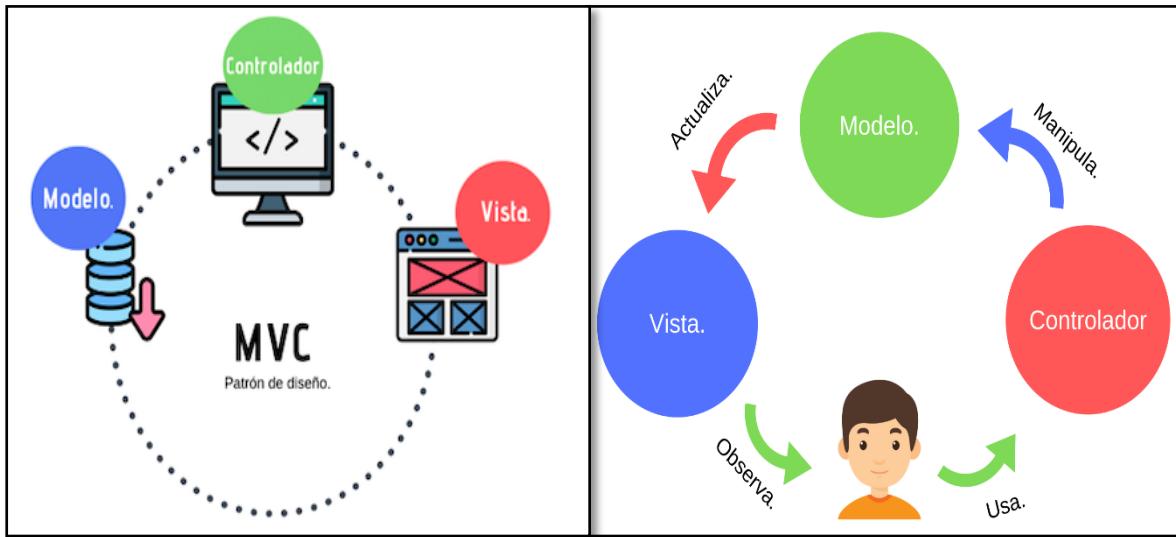


Figura N° 85: Modelo Vista Controlador - (MVC)
 (Cobian, 2019)

9.2. Patrón de Arquitectura por Capas

Este patrón es una de las técnicas más comunes que los arquitectos de software utilizan para dividir en partes más pequeñas un sistema de software. Al pensar en un sistema en términos de capas, se imaginan los principales subsistemas de software ubicados de la misma forma que los pisos de un edificio, donde cada capa (piso) descansa sobre la inferior. Por lo que la capa más alta utiliza varios servicios definidos por la inferior, pero la última es inconsciente de la superior; Además, normalmente cada capa oculta las capas inferiores de las siguientes superiores a esta. por ejemplo, en el edificio el primer piso tiene todo el circuito y tableros eléctricos que alimentan el edificio completo., pero la última es inconsciente de la superior.

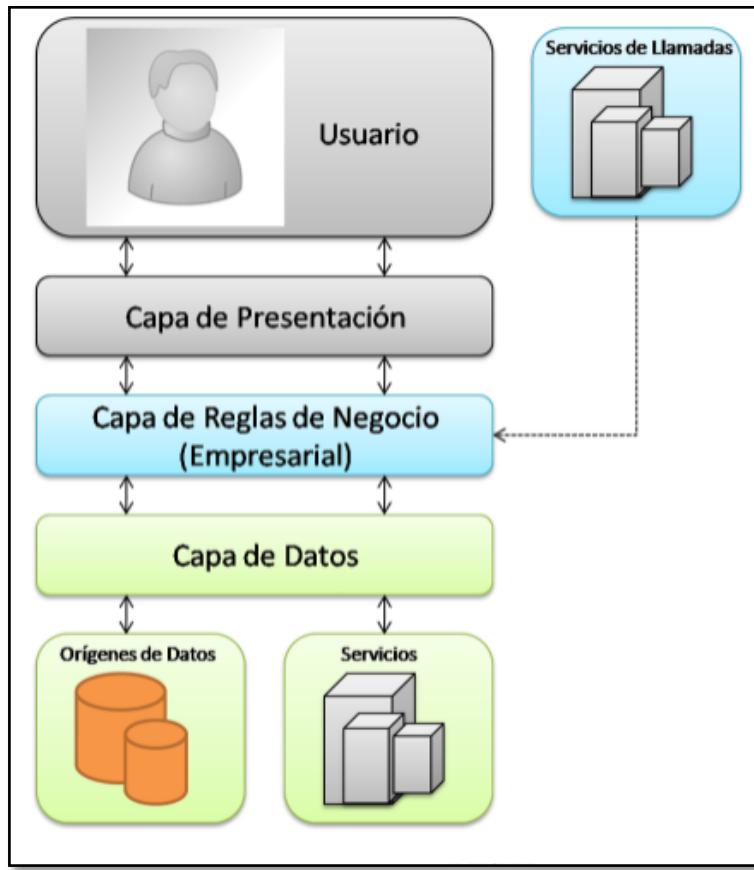


Figura N° 86: Esquema de Arquitectura por Capas
 (Cobian, 2019)

Capas principales de un patrón de arquitectura por capas:²⁰

1. Capa de Presentación: Referente a la interacción entre el usuario y el software. Puede ser tan simple como un menú basado en líneas de comando o tan complejo como una aplicación basada en formas. Su principal responsabilidad es mostrar información al usuario, interpretar los comandos de este y realizar algunas validaciones simples de los datos ingresados.

2. Capa de Reglas de Negocio (Empresarial): También denominada Lógica de Dominio, esta capa contiene la funcionalidad que implementa la aplicación. Involucra cálculos basados en la información dada por el usuario y datos almacenados y validaciones. Controla la ejecución de la capa de acceso a datos y servicios externos. Se puede diseñar la lógica de la capa de negocios para uso directo por parte de componentes de presentación o su encapsulamiento como servicio y

²⁰ patrón Arquitectura por capas: <https://arevalomaria.wordpress.com/2010/12/02/introduccion-al-patron-de-arquitectura-por-capas/>

llamada a través de una interfaz de servicios que coordina la conversación con los clientes del servicio o invoca cualquier flujo o componente de negocio.

3. Capa de Datos: Esta capa contiene la lógica de comunicación con otros sistemas que llevan a cabo tareas por la aplicación. Estos pueden ser monitores transaccionales, otras aplicaciones, sistemas de mensajería, etc. Para el caso de aplicaciones empresariales, generalmente está representado por una base de datos, que es responsable por el almacenamiento persistente de información. Esta capa debe abstraer completamente a las capas superiores (negocio) del dialecto utilizado para comunicarse con los repositorios de datos (PL/SQL, Transact-SQL, etc.).

9.3. Patrón de Repositorio²¹

“Un repositorio realiza las tareas de intermediario entre las capas de modelo de dominio y nuestros datos, actuando de forma similar a una colección en memoria de objetos del dominio...”.

Detallando un poco más la definición, diríamos que los repositorios son clases/componentes que encapsulan la lógica requerida para acceder a las fuentes de datos de la aplicación. Centralizan por lo tanto funcionalidad común de acceso a datos de forma que la aplicación pueda disponer de un mejor mantenimiento y desacoplamiento entre la tecnología y la lógica de las capas.

Enfoques de uso²²

Uno por cada modelo de negocio; la forma más directa es crear un repositorio para cada modelo de negocio clase. Pero cuidado que lo fácil puede complicarse, ya que puede conducir a problemas, como el código duplicado o complejidad, cuando varios repositorios necesitan interactuar entre sí.

El uso de la raíz de agregado; una raíz agregada es una clase que puede existir por sí mismo y es responsable de la gestión de las asociaciones con otras clases relacionadas. Por ejemplo, en una aplicación de comercio electrónico, usted tendría un OrderRepository que

²¹ Patrones de Diseño de software: <https://medium.com/@maniakhitocori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b>

²² El Patrón Repositorio. Enfoques de uso (n.d.). Azurewebsites.Net. Retrieved March 23, 2021, from http://jcarlosqp.azurewebsites.net/Blog/Post/patron_repositorio

se ocuparía de la creación de una orden y sus elementos de detalle de la orden correspondiente.

El Repositorio genérico; en lugar de crear clases de repositorios específicos, un desarrollador puede aprovechar de los genéricos .NET para construir un repositorio común que se puede utilizar a través de múltiples aplicaciones.

9.4. Patrón Cliente-Servidor

Este patrón consiste en dos partes; un servidor y múltiples clientes²³. El componente del servidor proporcionará servicios a múltiples componentes del cliente. Los clientes solicitan servicios del servidor y el servidor proporciona servicios relevantes a esos clientes. Además, el servidor sigue escuchando las solicitudes de los clientes.

Uso: Aplicaciones en línea como correo electrónico, uso compartido de documentos y banca.

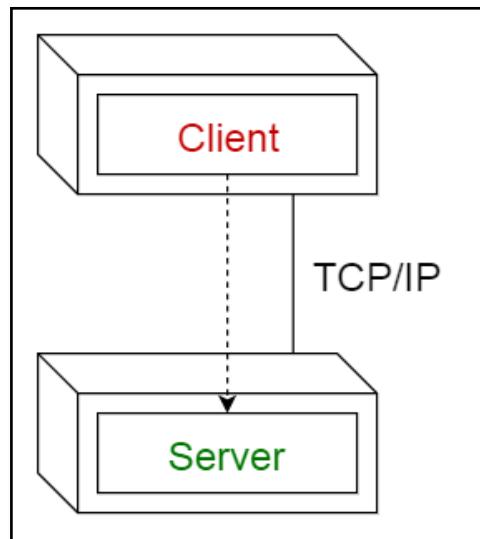


Figura N° 87 Patrón Cliente- Servidor
(huaman, 2018)

²³ Patrones de diseño: <https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b>

Componentes del modelo Cliente-Servidor

El modelo de arquitectura Cliente-Servidor, identifica cinco componentes que permiten articular dicha arquitectura, considerando que toda aplicación de un sistema de información está caracterizada por:

- Presentación/Captación de la información.
- Procesos.
- Almacenamiento de la información.
- Puestos de trabajo
- Comunicaciones.

Este modelo es bastante complejo; no obstante, esta es una forma de simplificar el funcionamiento la actuación de un sitio web frente a peticiones de usuario y poder entender la labor del hosting de un servidor enviando las respuestas. La arquitectura cliente-servidor tiene muchas ventajas como la de proporcionar una interfaz o servir de entorno laboral en distintas áreas del mercado.

Actividad del Aprendizaje Esperado N° 9

a) Relacione las arquitecturas con su definición correspondiente:

① <i>Diseño de software de arquitectura distribuida</i>	<input type="checkbox"/> <i>Debe responder al ámbito del problema en un tiempo dictado por el ámbito del problema, debe operar bajo restricciones de tiempo muy rigurosas.</i>
② <i>Diseño de software de arquitectura de tiempo real</i>	<input type="checkbox"/> <i>Es un sistema en el que el procesamiento de información se distribuye sobre varias computadoras</i>

c) Una las categorías de patrones con sus características correspondientes:

① *Antipatrón de Diseño*

Expresan un esquema organizativo estructural

② *Patrones de Diseño*

Esquemas para definir estructuras de diseño o sus relaciones

③ *Dialectos*

Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto

④ *Patrones de Arquitectura*

Dar a conocer los problemas que acarrean ciertos diseños muy frecuentes, para evitar cometer los mismos errores

d) Relacione las características de las arquitecturas descomposición modular y patrones de diseño correspondiente:

① *Patrones de Diseño*

Tamaño Pequeño

Abstracción

Independencia Modular

Ser Reutilizable

Comprobar su Efectividad

Encapsulamiento

② *Descomposición Modular*

Solución Actividad del Aprendizaje Esperado N° 1

Características	Tipos de Sistemas						
	TPS	OAS	MIS	DSS	AI	GDSS	ESS
<i>Sistema de Ventas y Marketing</i>	X						
<i>Sistema de Base de datos (Access)</i>		X					
<i>Sistema de Control de Inventario</i>			X				
<i>Sistema de pronóstico de presupuesto a cinco años</i>							X
<i>sistema de apoyo clínico que podría identificar el cáncer en etapas tempranas.</i>					X		
<i>Sistemas de Manufactura y Producción</i>	X						
<i>Sistema de Análisis de fijación de precios y rentabilidad</i>				X			
<i>Sistemas de Finanza y Contabilidad</i>	X						
<i>Software de diseño y dibujo industrial (AutoCAD)</i>		X					
<i>Sistemas de Recursos Humanos</i>	X						
<i>Sistema de para el Análisis de Reubicación de Personal</i>			X				
<i>Sistema de Reserva de asientos para una línea aérea</i>	X						
<i>Sistema de Pronostico de tendencia de ventas en periodos anuales</i>							X
<i>Sistema de planificación de recursos (Gantt Project)</i>		X					
<i>Sistema de Programación de la Producción</i>				X			
<i>Sistema de Votación Online</i>						X	
<i>Sistema para el diagnóstico de infecciones bacterianas.</i>					X		

Solución Actividad del Aprendizaje Esperado N° 2

Describa brevemente qué significan los siguientes términos:

- a) **Metodología de desarrollo de software.**

Una metodología de desarrollo de software se refiere a un framework (entorno o marco de trabajo) que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. ... Herramientas, modelos y métodos para asistir al proceso de desarrollo de software.

- b) **Modelo de Ciclo de Vida para el desarrollo de software.**

Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software, intenta determinar el orden de las etapas involucradas y los criterios de transición asociados entre estas etapas. Describe las fases principales de desarrollo de software.

- c) **Etapa dentro de un ciclo de vida.**

Las cuatro etapas de ciclo de vida son: Introducción, Crecimiento, Madurez y Declive. Todos los productos tienen un ciclo de vida y el tiempo en cada etapa varía de producto en producto.

- d) **Rol que puede cumplir una persona en el desarrollo de software.**

Estos roles son administradores de proyecto, analista, diseñador, programador, Tester, asegurador de calidad, documentador, ingeniero de manutención, ingeniero de validación y verificación, administrador de la configuración y, por último, el cliente.

- e) **Modelo/diagrama de las características de un sistema de software y sus partes componentes.**

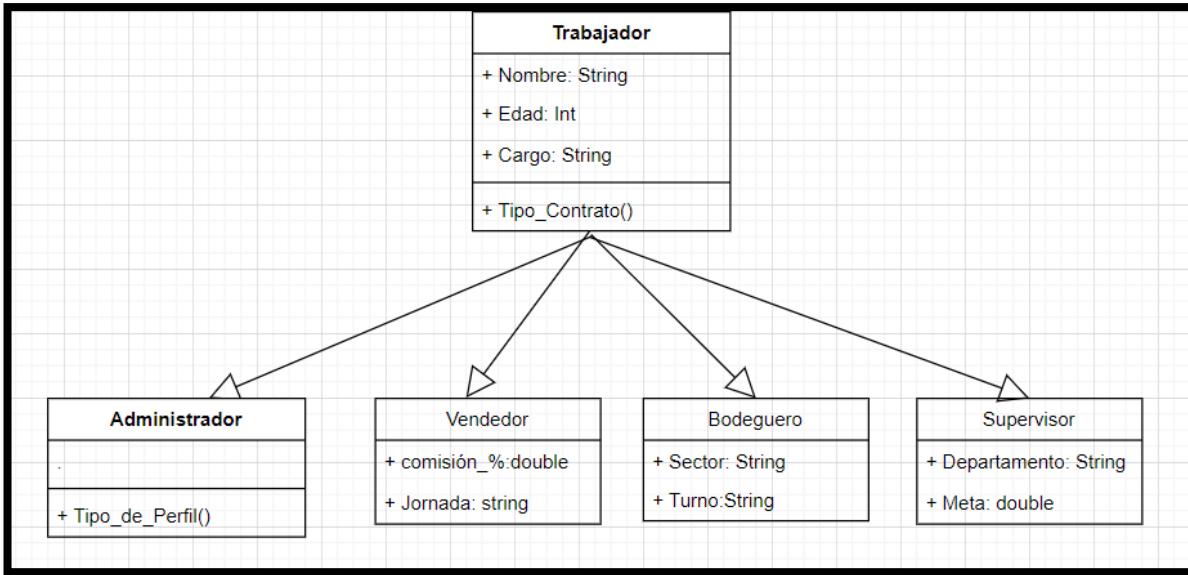
Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. ... Cada diagrama describe un apartado del sistema.

Solución Actividad del Aprendizaje Esperado N° 3

Pseudocódigo	Diagrama de Flujo
<p>1. <i>Inicio</i></p> <p>2. <i>Iniciar Variable</i> $PROMEDIO=0$</p> <p>3. <i>Solicitar introducción de valor NOTA</i></p> <p>4. <i>Leer NOTA</i></p> <p>5. <i>Almacenar en la variable PROMEDIO</i></p> <p>6. <i>Si PROMEDIO < 4.0 Entonces</i></p> <p>7. <i>Escribir PROMEDIO "Reprobado"</i></p> <p>8. <i>Sino</i></p> <p>9. <i>Si PROMEDIO ≥ 5.5 Entonces</i></p> <p>10. <i>Escribir PROMEDIO "Eximido"</i></p> <p>11. <i>Sino</i></p> <p>12. <i>Escribir PROMEDIO "Rinde Examen"</i></p> <p>13. <i>Fin_Si</i></p> <p>14. <i>Fin_Si</i></p> <p>15. <i>Fin</i></p>	<pre> graph TD Inicio[Inicio] --> Input["Introduzca valor NOTA"] Input --> Calculo[PROMEDIO = NOTA] Calculo --> Decision1{PROMEDIO < 4.0} Decision1 -- NO --> Decision2{PROMEDIO ≥ 5.5} Decision1 -- SI --> Reprobado["PROMEDIO \"REPROBADO\""] Decision2 -- NO --> Rinde["PROMEDIO \"RINDE EXAMEN\""] Decision2 -- SI --> Eximido["PROMEDIO \"EXIMIDO\""] Eximido --> FIN[FIN] Rinde --> FIN </pre>

Solución Actividad del Aprendizaje Esperado N° 4

Solución al problema planteado:



Solución Actividad del Aprendizaje Esperado N° 5

Resuelva el siguiente Crucigrama basado en el Aprendizaje Esperado N° 5:

Verticales

- 2:** Describe los elementos que componen un sistema. Debe detallar los elementos o componentes, las interacciones y relaciones, así como las interfaces públicas.
- 4:** Muestra la arquitectura de ejecución de un sistema. Incluye nodos, entornos de hardware y software.
- 7:** Representa los procesos de negocio o la lógica de un sistema complejo. Incluye, opcionalmente, el flujo de datos. El nivel de abstracción suele ser bastante alto, pero pueden realizarse diagramas de actividad exploratorios cuando la lógica que se trata es compleja.
- 8:** Muestra la secuencia de la lógica, el orden en que se suceden los mensajes. Importante, especialmente cuando se trabaja en ambientes altamente compartidos.
- 10:** Describe los estados de un objeto, así como la transición entre estados.
- 12:** Muestra una colección de clases, sus tipos, sus contenidos y sus relaciones. Importantísimo representa el modelo de datos, y en consecuencia su persistencia en alguna forma de almacenamiento.

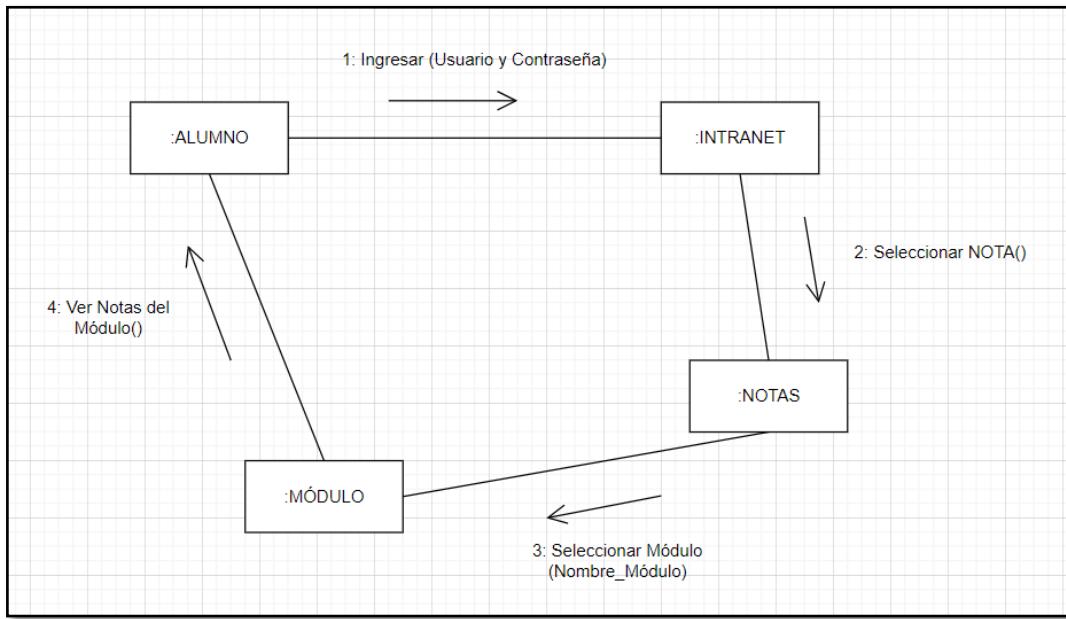
Horizontales

- 1:** Muestra las relaciones entre instancias de las clases y el flujo de mensajes entre ellas.
- 3:** Muestra un panorama general del flujo de control dentro del sistema o proceso de negocio.
- 5:** Muestra casos de uso individuales, actores y las relaciones entre ellos. El Proceso Unificado dice está dirigido por los casos de uso, esto significa que este diagrama (en el nivel de abstracción que sea) es la base del lenguaje de modelado y representación.
- 6:** Muestra la estructura interna de una clase, componente o caso de uso. Especialmente debe indicar los puntos de interacción con otras partes del sistema.
- 9:** Describe como los elementos del modelo se organizan en \"paquetes\", debe indicar la dependencia entre paquetes.
- 11:** Describe los objetos y sus relaciones en algún momento. Generalmente se usa en casos especiales para diagramas de clase o de comunicaciones.
- 13:** Muestra el cambio de estado de un objeto a través del tiempo en respuesta a eventos externos.

	D	I	A	G	R	A	M	A	D	E	C	A	S	O	S	D	E	U	S	O	
							I														
						D	I	A	G	R	A	M	A	D	E	E	S	T	R	U	
						I	G										C	T	U	R	
			D	A	R	D	I	A	G	R	A	M	A	D	E	C	O	M	U	N	
			D	A	R	M	I	A	G	R	A	M	A	D	E	C	O	M	U	N	
	D	A	R	M	D		D	I	A	G	R	A	M	A	D	E	O	B	J	E	
D	I	A	G	R	A	M	A	D	E	I	T	E	R	A	C	C	I	Ó	N		
A	R	M	D	D																	
G	A	A	E	E																	
R	M	D	C	S																	
A	A	E	O	P																	
M	D	S	M	L																	
A	E	E	P	I																	
D	A	C	O	E																	
E	C	U	N	G																	
E	T	E	E	U	D	I	A	G	R	A	M	A	D	E	T	I	E	M	P	O	
S	I	N	N	E	I	I	A	G	R	A	M	A	D	E	T	I	E	M	P	O	
T	V	C	T																		
A	I	I	E																		
D	D	A	S																		
O	A																				
S	D	I	A	G	R	A	M	A	D	E	P	A	Q	U	E	T	E	S			

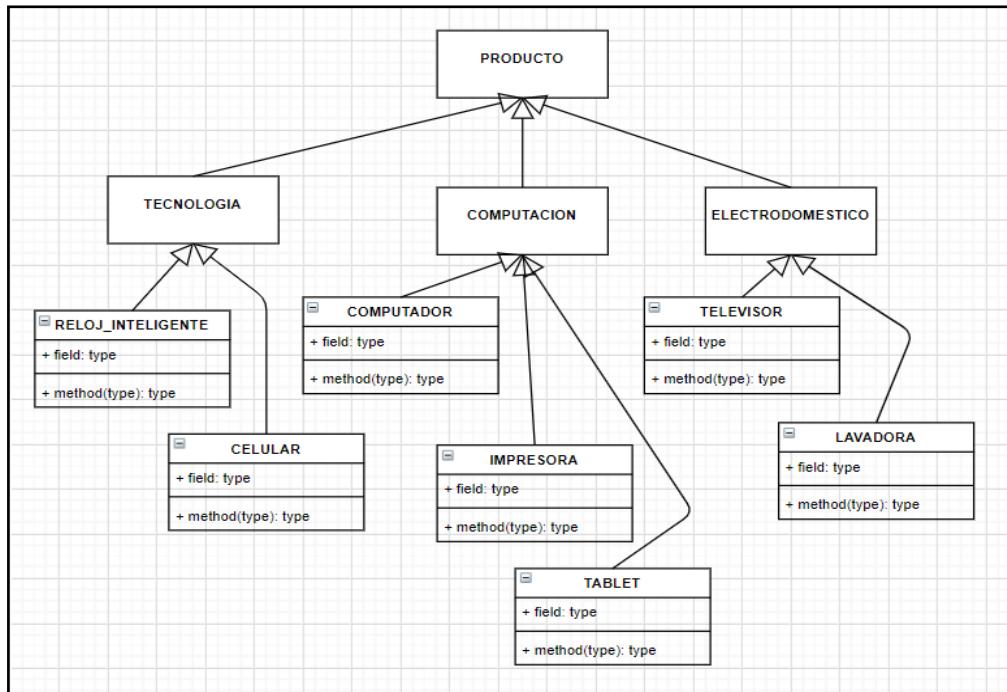
Solución Actividad del Aprendizaje Esperado N° 6

Resultado del diagrama de colaboración para el ejemplo entregado:



Solución Actividad del Aprendizaje Esperado del Aprendizaje Esperado N° 7

Resultado del diagrama de clase para el ejemplo entregado:



Solución Actividad del Aprendizaje Esperado N° 8**1. ¿Qué es la Arquitectura de Software?**

- A) Es la manera en la que un componente no se relaciona en una estructura
- B) Ninguna de las anteriores
- C) Es la manera en la que un componente se relaciona con uno similar
- D) Es la manera en la que los distintos componentes se relacionan para crear una estructura

2. ¿Qué es una descripción arquitectónica?

- A) Es un conjunto de productos externos de un trabajo terminado que no reflejan puntos de vistas de un sistema.
- B) Ninguna de las anteriores
- C) Es un conjunto de productos de trabajo que reflejan puntos de vista distintos del sistema
- D) Son partes separadas de un producto terminado que no dan puntos de vista y se elaboran independientemente

3. ¿Qué permite la representación de la Arquitectura de Software?

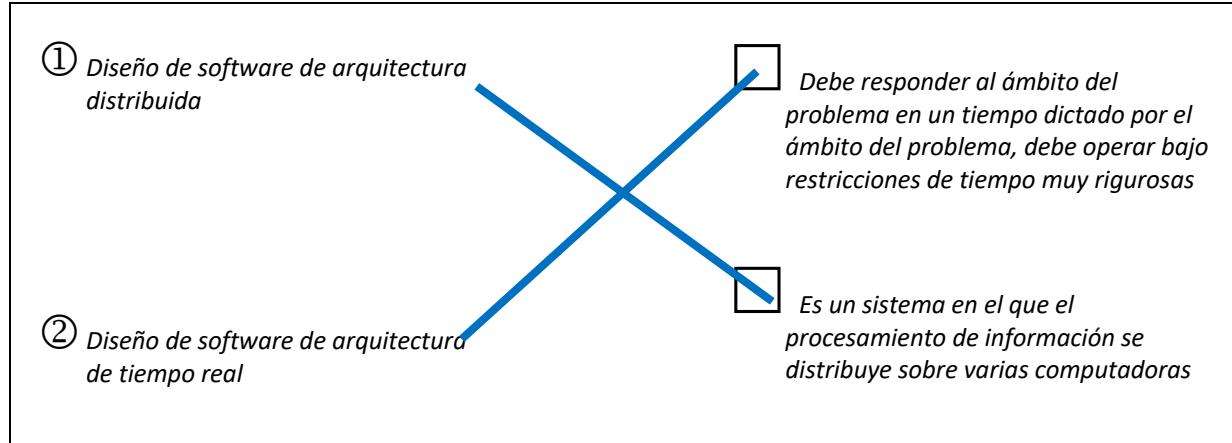
- A) Representan el desarrollo de estructuras multinacionales
- B) Crean estructuras para la elaboración de un sistema de software
- C) Permiten la comunicación entre todas las partes
- D) Permiten la comunicación entre una parte solamente

4. Es el nivel en el que el arquitecto usa un diagrama de contexto

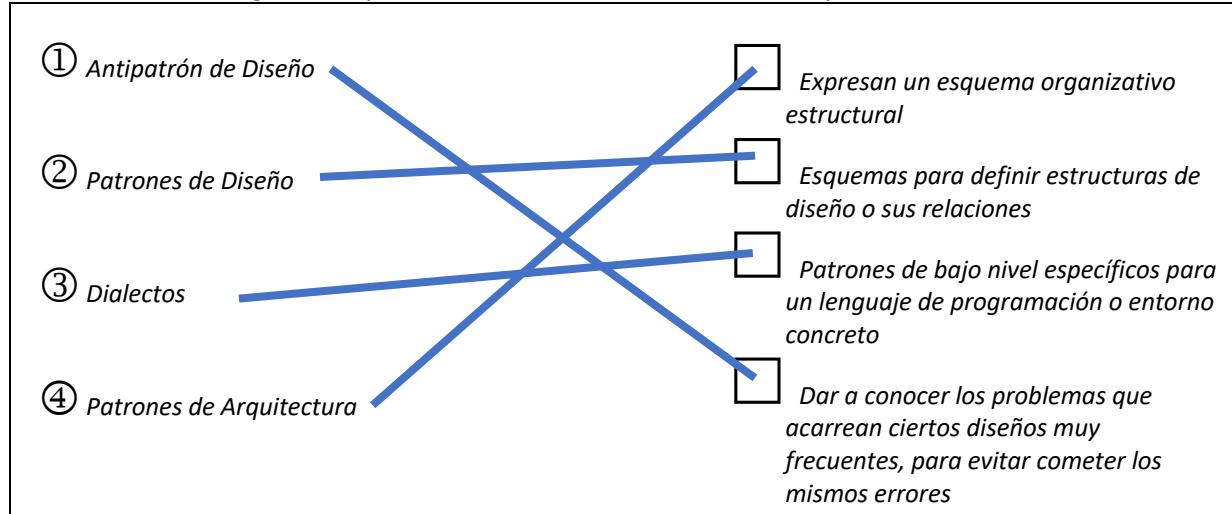
- A) Nivel de diseño Arquitectónico
- B) Nivel de Estructura
- C) Nivel de Liderazgo

Solución Actividad del Aprendizaje Esperado N° 9

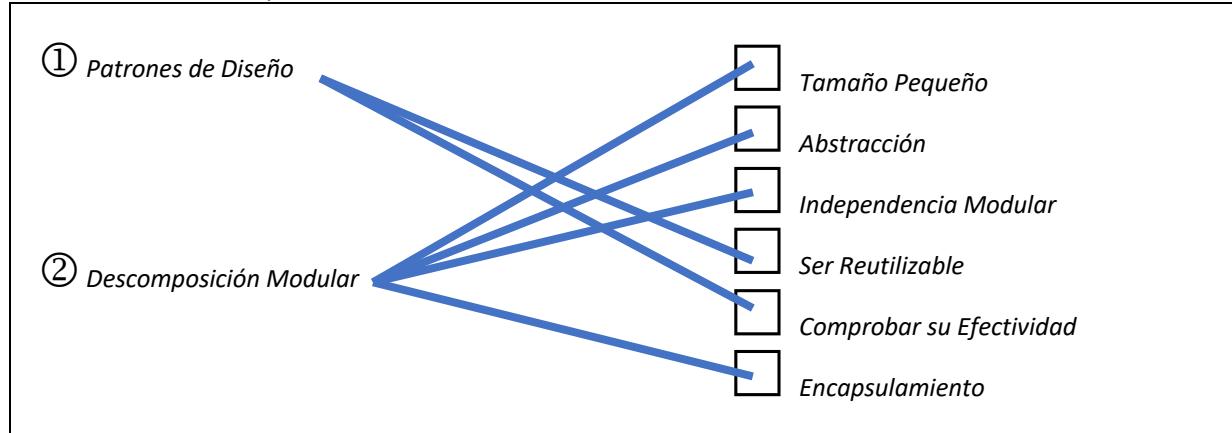
a) Relacione las arquitecturas con su definición correspondiente:



b) Una las categorías de patrones con sus características correspondientes:



c) Relacione las características de las arquitecturas descomposición modular y patrones de diseño correspondiente:



Bibliografía

Guía de ingeniería del software, Instituto Nacional de Tecnologías de la Comunicación de España (INTECO) Pág. 30-32.

(EKCIT, C. E. (s.f.). *Tic. Portal CRM*. Obtenido de <https://www.ticportal.es/glosario-tic/crm-gestion-relaciones-cliente>

Aner Technology. (s.f.). Obtenido de <https://www.aner.com/que-es-un-erp.html>

Apd. (s.f.). Obtenido de <https://www.apd.es/niveles-gestion-empresarial/>

Bentley, L. D., & Whitten, J. L. (2008). Parte Dos Análisis de Sistemas. En *Análisis de Sistemas: Diseño y Métodos 7º Edición*. McGraw-Hill.

Bizneo Blog. (s.f.). Obtenido de <https://www.bizneo.com/blog/estructura-organizacional/>

cemeblog. (s.f.). Obtenido de <https://blog.cemebe.info/project-cartoon/>

Cobian, V. G. (Mayo de 2019). Obtenido de <https://blog.nearsoftjobs.com/patr%C3%B3n-de-dise%C3%BAo-mvc-2366948b5fc7>

Diagramas UML. (s.f.). Obtenido de <https://diagramasuml.com/despliegue/>

Digital Guide IONOS. (2020). Obtenido de <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/codigo-fuente-definicion-con-ejemplos/>

EKCIT, I. d. (s.f.). *Tic.Portal/ERP*. Obtenido de <https://www.ticportal.es/temas/enterprise-resource-planning/que-es-sistema-erp>

Empiria. (s.f.). Obtenido de <https://www.redalyc.org/pdf/2971/297124024004.pdf>

Eni Ediciones. (s.f.). Obtenido de <https://www.ediciones-eni.com/open/mediabook.aspx?idR=71fee384bc83826a4494aed0a7206cdc>

Fuentes, M. d. (2011). *Notas del Curso Análisis de Requerimientos*. México: Universidad Autónoma Metropolitana.

Hernández, R. R. (s.f.). *Programación Orientada a Objetos con C#*. Obtenido de <http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/POO/Apuntes/04.-%20Polimorfismo.pdf>

Huete-Huerta, S. L.-N. (s.f.). Obtenido de <https://diagramasuml.com/diagrama-de-clases/>

INEI. (s.f.). Obtenido de <https://es.slideshare.net/tomaspetto/herramientas-case-7275728>

Kaptadata. (2017). Obtenido de <https://kaptadata.com/que-es-business-intelligence/>

Kendall, K. E., & Kendall, J. E. (2011). Parte I: Fundamentos del Análisis de Sistemas. En *Análisis y Diseño de Sistemas 8 Edición*. Pearson.

ManagementPro. (s.f.). Obtenido de <https://blog.mproerp.com/que-es-un-sistema-de-punto-de-venta/>

Mediavilla, E. (s.f.). *Programación Orientada a Objetos*. Obtenido de https://www.ctr.unican.es/asignaturas/MC_OO/Doc/Casos_de_uso.pdf

Pressman Roger S. (2005). *Ingeniería del Software: Un enfoque Práctico*.

Pressman, R. S. (2010). Ingeniería del Software Un Enfoque Práctico. En *Capítulo 3 Desarrollo Ágil* (págs. 69-70). México: Mc Graw-Hill.

Propia, Elaboración. (s.f.).

RedHat. (s.f.). Obtenido de <https://www.redhat.com/es/topics/cloud-computing/what-is-iaas>

SmartDraw. (s.f.). Obtenido de <https://www.smartdraw.com/flowchart/simbolos-de-diagramas-de-flujo.htm>

Sparx Systems. (s.f.). Obtenido de http://www.sparxsystems.com.ar/resources/tutorial/physical_models.php

Tecnologías Información. (2018). Obtenido de <https://www.tecnologias-informacion.com/ciclovida.html#>

Towers, A. (s.f.). Obtenido de https://www.academia.edu/11030132/Propuestas_para_el_mejoramiento_de_la_calidad_del_Diagrama_de_Clases_de_UML

Villalobos Abarca, M. (2019). *Proyectos de Desarrollo de Software (VERSIÓN 1.0)*. Arica, Chile.

Tablas, esquemas

Figura N° 1: Elementos de un Sistema.....	3
Figura N° 2: Tipos de Sistemas de Software	5
Figura N° 3: Contexto del Análisis de Sistemas.....	6
Figura N° 4: Tipos de Proyectos	8
Figura N° 5: Procesos de Gestión para un Proyecto	9
Figura N° 6: Ciclo de Vida de Sistemas	17
Figura N° 7: Las 7 Fases del ciclo de desarrollo de sistemas (SDLC)	17
Figura N° 8: Problemáticas del desarrollo de software	20
Figura N° 9: Criterios de los Objetivos del Ciclo de Vida.....	21
Figura N° 10: Flujo de Proceso Scrum.....	22
Figura N° 11: Elementos de Scrum	23
Figura N° 12: El ciclo de liberación de la programación extrema	23
Figura N° 13: Los cinco valores de XP	24
Figura N° 14: El ciclo de vida clásico: Modelo "en cascada"	25
Figura N° 15: Ventajas y Desventajas del Modelo Cascada	Error! Bookmark not defined.
Figura N° 16: Modelo Incremental	27
Figura N° 17: Ventajas y Desventajas del Modelo Incremental.....	Error! Bookmark not defined.
Figura N° 18: Modelo Iterativo	28
Figura N° 19: Ventajas y Desventajas del Modelo Iterativo	Error! Bookmark not defined.
Figura N° 20: Estructura Organizacional.....	30
Figura N° 21: Elementos básicos de las Organizaciones.....	31
Figura N° 22: Gráfico de datos de «Microsoft Dynamics Enterprise Applications for SMB».....	32
Figura N° 23: Tipos de ERP.....	33
Figura N° 24: Ventajas y Desventajas de los ERP.....	Error! Bookmark not defined.
Figura N° 25: Niveles de Organización Empresarial o Administrativa	34
Figura N° 26: Ventajas de la gestión de relaciones con el cliente	36
Figura N° 27: Prioridades del CRM dependiendo del tipo de relación	36
Figura N° 28: Flujo de información entre departamentos con CRM	37
Figura N° 29: BI a Nivel Organizacional	38
Figura N° 30: Beneficios de Business Intelligence	39
Figura N° 31: Características de Business Intelligence	39
Figura N° 32: Componentes de BI.....	40
Figura N° 33: Etapas de Implementación de BI	41
Figura N° 34: Sistemas POS Tradicionales	Error! Bookmark not defined.
Figura N° 35: Sistema POS en la Nube	Error! Bookmark not defined.
Figura N° 36: Elementos que integran un Punto de Venta	44
Figura N° 37 Servicios de SaaS.....	46
Figura N° 38: Símbolos del Diagrama de Flujo.....	48

Figura N° 39: Símbolos más utilizados en DF.....	49
Figura N° 40: Símbolos más utilizados en DF (continuación).....	50
Figura N° 41: Elementos del Diagrama de Flujo de Datos	52
Figura N° 42: Tipo de Entidad y sus Atributos	54
Figura N° 43: Diagrama de Clases de un sistema de Aviación	56
Figura N° 44: Relación Reflexiva	57
Figura N° 45: Ejemplo de Relación.....	57
Figura N° 46: Asociación	58
Figura N° 47: Agregación	58
Figura N° 48: Composición	59
Figura N° 49: Dependencia	59
Figura N° 50: Herencia o Generalización de clases.....	60
Figura N° 51: Ejemplo de Generalización	61
Figura N° 52: Ejemplo de Polimorfismo.....	62
Figura N° 53: Representación de Polimorfismo.....	63
Figura N° 54: Clasificación de Polimorfismo	63
Figura N° 55: Ejemplo jerarquía de Clases.....	64
Figura N° 56: Diagrama de Clases de UML, Mejorado.....	65
Figura N° 57: Una Clase y sus atributos	66
Figura N° 58: Lista de métodos de la Clase Lavadora	66
Figura N° 59: Instancias de Clases	68
Figura N° 60: Diagrama de Implementación.....	72
Figura N° 61: Diagrama de Casos de Uso UML	72
Figura N° 62: Relación Include.....	74
Figura N° 63: Relación Extend	74
Figura N° 64: Un diagrama de actividades en el que se muestra la manera en que actor camina ..	75
Figura N° 65: Diagrama de Actividad con carriles.....	75
Figura N° 66: Diagrama de Secuencia.....	76
Figura N° 67: Diagrama de Clases de un sistema de aviación.....	77
Figura N° 68: Modelo de Casos de Uso.....	80
Figura N° 69: Diagrama de Paquetes	81
Figura N° 70: Vista de Despliegue.....	82
Figura N° 71: MF01 Modelo Físico.....	83
Figura N° 72: Nodo y sus propiedades.....	83
Figura N° 73: Símbolo que representa un componente	84
Figura N° 74: Adición de información al símbolo del componente	85
Figura N° 75: Símbolos de relaciones entre componentes y las clases	85
Figura N° 76: Ejemplo de diagrama de despliegue	88
Figura N° 77: Proceso de modelado con abstracción intermedia	89
Figura N° 78: Herramientas CASE en base a las fases que cubren	92
Figura N° 79: App Studio	96

Figura N° 80: Código Fuente	101
Figura N° 81: Diseño como caja negra (las entradas están a la izquierda, las salidas, a la derecha, y los responsables, arriba).....	104
Figura N° 82: Proceso general de diseño de la arquitectura.	105
Figura N° 83: Modelo Vista Controlador - (MVC)	113
Figura N° 84: Esquema de Arquitectura por Capas	114
Figura N° 85 Patrón Cliente- Servidor.....	116