Técnicas para formular proyectos de software e ingeniería web

Ángel Alberto Varón Quimbayo

Técnicas para formular proyectos de software e ingeniería web

Ángel Alberto Varón Quimbayo

AREANDINA
Fundación Universitaria del Área Andina

Varón Quimbayo, Ángel Alberto / autor
Técnicas para formular proyectos de software e ingeniería web -- / autor Ángel Alberto Varón Quimbayo -- Bogotá: Fundación Universitaria del Área Andina, 2021.
ISBN (digital): 978-958-5139-39-8
105 páginas : gráficos , tablas ; 28 cm.
Incluye índice
.
1. Elaboración de proyectos. – Desarrollo ágil de software. – 2. Ingeniería de línea de productos de software. – 3. Proyectos de desarrollo.

Catalogación en la publicación Biblioteca Fundación Universitaria del Área Andina (Bogotá)
005.1 – scdd22

TÉCNICAS PARA FORMULAR PROYECTOS DE SOFTWARE E INGENIERÍA WEB

ISBN (digital): 978-958-5139-39-8

FUNDACIÓN UNIVERSITARIA DEL ÁREA ANDINA

Calle 70 No. 12-55, Bogotá, Colombia

Tel: +57 (1) 7424218 Ext. 1231

Correo electrónico:
publicaciones@areandina.edu.co

PROCESO EDITORIAL

Dirección editorial:
Omar Eduardo Peña Reina

Coordinación editorial:
Camilo Andrés Cuéllar Mejía

Diagramación y armada interior y cubierta:
Proceditor
Calle 1C No. 27A-01, Bogotá, Colombia
Tel.: 757 9200
Correo electrónico:
proceditor@yahoo.es

BANDERA INSTITUCIONAL BOGOTÁ
BANDERA INSTITUCIONAL

Pablo Oliveros Marmolejo ǂ
Gustavo Eastman Vélez
Miembros Fundadores

Diego Molano Vega
Presidente de la Asamblea General y Consejo Superior

José Leonardo Valencia Molano
Rector Nacional y Representante Legal

Martha Patricia Castellanos Saavedra
Vicerrectora Nacional Académica

Ana Karina Marín Quirós
Vicerrectora Nacional de Experiencia Areandina

Karol Milena Pérez Calderón
Vicerrectora Nacional de Crecimiento y Desarrollo

Erika Milena Ramírez Sánchez
Vicerrectora Nacional Administrativa y Financiera

Felipe Baena Botero
Rector - Seccional Pereira

Gelca Patricia Gutiérrez Barranco
Rectora - Sede Valledupar

María Angélica Pacheco Chica
Secretaria General

Omar Eduardo Peña Reina
Director Nacional de Investigaciones

Cristian Julián Díaz Álvarez
Decano Facultad de Ingenierías y Ciencias Básicas

Julio Alberto Castillo Ramírez
Director programa Ingeniería de Sistemas

Camilo Andrés Cuéllar Mejía
Subdirector Nacional de Publicaciones

Ángel Alberto Varón Quimbayo

Magíster en Gestión Diseño y Dirección de Proyectos por la Universidad Internacional Iberoamericana de Puerto Rico, Especialista en Gestión Pública por la Escuela Superior de Administración Pública (esap) del Tolima, Ingeniero de Sistemas, Fundación Universitaria del Área Andina, Colombia.

Lista de figuras

Lista de tablas

Formulación de proyectos

1

De acuerdo con la guía PMBOK (PMI, 2008), un proyecto es un esfuerzo temporal que se lleva a cabo con el fin de crear un producto, un servicio o un resultado único.

Un proyecto también es el desempeño durante un periodo de tiempo que se realiza para crear un beneficio, obtener una ganancia o un provecho, y se especifica como un conjunto coherente e integral de actividades y tareas tendientes a alcanzar un objetivo que contribuya a la solución de un problema, una parte de un problema o una necesidad, en un periodo de tiempo determinado.

Estas acciones se expresan en documentos en los que se plantea y describe detalladamente cada una de las actividades que se van a desarrollar. En el caso del desarrollo de *software* o de aplicaciones web, se parte de una base teórica, conceptual y metodológica en la que se incluyen los recursos humanos, técnicos, tecnológicos y financieros requeridos para la realización de cada una de las labores. En este proceso se escoge un modelo o una metodología, se plantea cada fase del desarrollo del *software* y se debe tener claridad y ser cuidadoso, ya que este tiene varios componentes a lo largo del ciclo de vida tales como análisis, diseño, programación y puesta en marcha, de modo que se desarrolla en función de la misión, la visión y los objetivos de la organización, dependiendo de los procesos, los procedimientos y las interacciones de esta.

En este capítulo se tratan temas básicos para la formulación del proyecto de *software* e ingeniería de web que nos sirven como carta de navegación en el momento de formular un proyecto de desarrollo, lo cual, sin duda, nos ayudará a agilizar la tarea y nos permitirá presentar la documentación mucho mejor estructurada, a fin de que sea posible evidenciar que conocemos del tema de lo que estamos haciendo y, a su vez, decidir hasta dónde llegar con la solución que planteamos y en qué tiempo lo vamos a lograr; lo más importante es saber el costo real de la puesta en marcha del producto a desarrollar, pues con esto se podrá determinar la viabilidad del proyecto y demostrar a nuestros clientes que la solución que les estamos planteando es adecuada y acorde a sus necesidades. Además, podremos confirmar la utilización de herramientas apropiadas para el desarrollo de la solución, así como los elementos más importantes de las organizaciones tales como el ciclo de Denimg, que se basa en PHVA y una buena planificación, la forma de hacerlo, verificar y actuar, todo esto con el propósito de neutralizar algunos de los problemas comunes que se presentan en el momento de sustentar nuestro proyecto (Project Management Institute, 2004).

Anteproyecto

La formulación de un proyecto de desarrollo de *software* o desarrollo web consiste en analizar una problemática u oportunidad dentro de una organización. Para esto debemos acudir al llamado de un gerente o un miembro que solicite nuestros servicios, o como lo hacen las grandes organizaciones, al participar en un proceso licitatorio o una convocatoria en la que nos dan a conocer las necesidades. A partir de esta debemos realizar un estudio minucioso y, según los resultados obtenidos, presentamos la propuesta o el anteproyecto, lo cual consiste en una documentación muy bien estructurada en la que plasmamos lo que se va a solucionar y hasta dónde debemos llegar, es decir, el alcance del proyecto. Este contiene la información preliminar que nos permitirá demostrar que emprender el proyecto si es viable para la empresa, sobre todo en la parte de costo beneficio, en la cual realizamos un estudio de factibilidad por el que se expresa la viabilidad financiera, técnica, operativa, ambiental y legal, entre otras.

En este primer capítulo se abordan algunos conceptos que nos explican los apartados que debe tener un anteproyecto:

- título
- planteamiento del problema
- objetivo del proyecto
- objetivos específicos
- justificación
- marco de referencia
- cronograma de actividades
- estudio de factibilidad

Ítems de un anteproyecto

Título

Es el nombre que se le da al proyecto y tiene como objetivo presentar en pocas palabras su contenido, ya que constituye la idea principal; la expone de forma breve, con palabras precisas que indiquen la naturaleza y el objetivo del proyecto. A partir del título es posible deducir de qué trata el contenido, puesto que manifiesta el asunto al que hace referencia. Es primordial elegir un buen título, ya que debe ser una herramienta potente y coherente que añada valor al proyecto (Gonzales, 2009).

Planteamiento del problema

Consiste en describir el problema de forma detallada, clara y sintética, de manera que se contextualice teórica y referencialmente. No obstante, debe haber objetividad ante la dificultad en el momento de sustentarla, ya que influye de manera decisiva en el desarrollo del proyecto. Esto nos lleva a determinar los elementos esenciales, como lo son, por ejemplo, la causa y el efecto, a fin de analizar su viabilidad. Al identificarlo se debe recolectar y analizar la mayor cantidad de información posible y combinar la exposición que nos han hecho los usuarios directa e indirectamente implicados (Tamayo, 2002).

Objetivo general del proyecto

Cada vez que emprendemos un proyecto lo hacemos para alcanzar algún objetivo. Es así que decidimos hasta dónde pretendemos llegar y a qué se aspira con él, razón por la cual se plantea un objetivo principal que realmente sustenta lo que anhelamos hacer. Este permite orientar la solución al problema o la necesidad, se considera el punto central o de partida del proyecto y describe el propósito por el que se va a llevar a cabo, qué es lo que se va a solucionar y hasta dónde se aspira a llegar, así como su finalidad y las metas a alcanzar. Se debe señalar cuál es y de qué manera se va a desarrollar al expresarlo de manera clara, concreta y precisa, a fin de que sirva como guía del proyecto a realizar. El ingrediente clave es que permite saber por qué haces las cosas y qué se quiere con cada tarea, por lo cual se debe trabajar con un objetivo concreto y claro (Tamayo, 2002).

Objetivos específicos

Se deben plantear varios objetivos específicos que sirvan como complemento del objetivo general, por tanto, deben de estar alineados y delimitados con este, pues estos se generan a medida que se avanza en el proceso del análisis del proyecto y con la evolución de cada fase, al traducir cada objetivo en una tarea concreta. Esto permite establecer orden, de modo que se logran transmitir los resultados que se desean obtener con la realización del proyecto.

Se podría decir que por cada objetivo específico se definirán resultados, al ser estos resultados niveles en los que nuestro proyecto designará responsabilidades. Así, se logra que los objetivos sean factibles, medibles y precisos, e identificar quiénes se beneficiarán y cuál es el impacto a generar; también deben ser realistas y alcanzables, es decir, deben existir los recursos y ser posible llevarlos a cabo en el tiempo establecido (Tamayo, 2002).

Justificación

Consiste en expresar por qué razón es conveniente llevar a cabo el proyecto y por qué es útil. La justificación debe efectuarse con un propósito bien definido, pues alude a las razones que llevaron a seleccionar el tema del proyecto, así como ser estas suficientemente

fuertes para que determinen su realización (esto determina la pertinencia del proyecto). Además, se debe indicar con fundamentos los motivos por los cuales el problema merece ser intervenido como potencialidad, necesidad y oportunidad del proyecto, desde el punto de vista organizacional, disciplinar y tecnológico (Gonzales, 2009).

Marco de referencia

El marco de referencia sirve como instrumento en la gestión y la ejecución del proyecto, presenta los lineamientos generales o las directrices definidas; es el extracto de varios componentes conceptuales que sirven de base para indagar antecedentes, bases teóricas y la definición de términos como elementos esenciales a tener en cuenta en el tiempo de duración del proyecto.

Lerma (2001) afirma al respecto: "El marco de referencia inscribe el problema a investigar dentro del conjunto de conocimientos, variables, conceptos, hipótesis y teorías desarrolladas por otros investigadores sobre el tema" (p. 44).

Cronograma de actividades

Para crear el cronograma de actividades del proyecto, primero, debemos identificar el tipo de producto a realizar: puede ser *software* de aplicación, de sistema o una aplicación con orientación a la web. Después de esto seleccionamos una metodología de desarrollo según el producto a obtener. Estos modelos de ciclo de vida de desarrollo constan de unas fases, las cuales, a su vez, se componen de actividades o tareas propias del modelo que se deben realizar según su estructura. A partir de esto y con ayuda de un calendario realizamos la planificación de cada una de las actividades en un lapso determinado de tiempo, para lo cual se tiene presente la fecha inicial y la fecha final; en esta actividad es recomendable utilizar herramientas digitales como, por ejemplo, Project, que es la más conocida; sin embargo, existen muchas *on line* y se puede seleccionar la que se considere más adecuada (Sommerville, 2005).

Estudio de factibilidad

El estudio de factibilidad consiste en demostrar la viabilidad del proyecto en diferentes aspectos, de manera que se debe demostrar que la inversión a realizar es pertinente según costo beneficio, para lo cual se revisan los aspectos que se enlistan y describen a continuación.

- *Viabilidad operativa*. Es el estudio de viabilidad operativa; p. ej., si vamos a desarrollar un software debemos tener presente el sistema operativo que se utiliza en la empresa con miras a que el producto se pueda implementar en este, las impresoras que se tienen, también revisamos los integrated development environment (IDE) a utilizar, la red de datos y los usuarios que van a utilizar la aplicación, a fin de determinar si con

una pequeña capacitación pueden operar el producto, porque, de lo contrario, se puede constituir en un aspecto negativo para esta viabilidad.

- *Viabilidad legal*. Consiste en realizar un análisis exhaustivo en el que verificamos los aspectos legales que debemos tener presentes para desarrollar el *software* y que este no afecte la organización; p. ej., si desarrollamos un *software* para nomina, debemos regirnos por el Código Sustantivo del Trabajo colombiano; si realizamos un producto web para comercio electrónico, nos acogemos a la legislación que nos exige la DIAN, la ley de derechos de autor, etc.
- *Viabilidad financiera*. Consiste en determinar el costo del producto, de modo que se debe tener en cuenta el recurso humano que va a participar en su desarrollo, el tiempo que va a durar el proyecto y la mano de obra de cada individuo. También se incluyen los recursos que se van a utilizar a lo largo del proyecto, tales como papelería, equipos de cómputo, licencias IDE, costos de reuniones, videobeam y capacitación, entre otros (Sommerville, 2005).

Administración de proyectos

2

La administración de proyectos es una labor complicada que requiere la coordinación en el tiempo de equipos de trabajo, proveedores, personas, tareas, actividades, insumos y costos. Requiere de una buena planificación y de mantener un balance entre costo, tiempo y calidad. Debe realizarse desde antes del inicio del proyecto apoyada en el uso de herramientas que automaticen y ayuden a la organización de este. Además, debe hacerse seguimiento y control, a fin de estar en capacidad de medir el progreso y velar por que las labores logren alcanzar la satisfacción completa de las necesidades de los usuarios. En este sentido, influye de manera constante una muy buena comunicación tanto interna como externa entre los clientes del proyecto, los usuarios, los proveedores y los gestores de la organización, a fin de que se atienda, realice y trabaje con un enfoque hacia la misión y la estrategia del proyecto. La comunicación debe aplicarse en todo el periodo de duración del proyecto y se puede generar diferentes estrategias con este propósito, tales como obtener información simultánea sobre la ejecución, reportar su estado, negociar y asignar trabajo, o socializar reportes y resultados, entre otros.

Conceptos sobre administración de proyectos

La administración, gestión o dirección de proyectos es una especialidad que se encarga de la administración de los recursos destinados a la ejecución del proyecto. En esta se deben aplicar cuatro conceptos fundamentales para alcanzar el propósito por el cual se va a desarrollar el *software*, los cuales se presentan en la Figura 1.

Conceptos fundamentales sobre gestión de proyectos

Los conceptos fundamentales sobre gestión de proyectos se enlistan y describen a continuación.

- *Inicio*. En el momento de dar inicio al proyecto de desarrollo de *software* la actividad principal es la recolección de información pertinente y necesaria, con el objetivo de realizar un análisis que permita establecer el alcance del proyecto a fin de saber cuáles son los productos a entregar y, en efecto, establecer los requerimientos para su buen funcionamiento. Es en este espacio en el que se debe identificar qué hace y qué no hace el sistema, así como establecemos el ambiente en el que este será implementado, las funcionalidades y, posiblemente, elaborar una imagen de su arquitectura, ya que se

hace necesario identificar el medio ambiente del *software*, es decir, el sistema operativo en el que se va a soportar. También se debe definir si es un *software* de escritorio o si es orientado a la web, con el fin de realizar la estimación de riesgos, el costo financiero y el tiempo de duración del proyecto. Todos estos postulados deben de ir consignados en el acta de inicio.

- *La planificación de proyectos*. En esta etapa es fundamental definir la metodología que se va a utilizar en el proceso de desarrollo del proyecto, todo para identificar las fases y cada una de las tareas a desarrollar. A partir de esta acción es que podemos realizar la programación de cada una de las labores, al asignar un espacio de tiempo, los responsables de cada actividad y los insumos que se requieren, de manera que se debe establecer el costo global. Un factor importante es la conformación del equipo de desarrollo, el equipo de aseguramiento de la calidad del software que garantizará la calidad del producto y el equipo que determinará las técnicas, las herramientas y las estrategias que permitan garantizar la seguridad del *software* una vez finalizado.
- *Seguimiento y control*. Es un procedimiento que debe hacerse a lo largo del proyecto, desde el momento en el que se da inicio al proyecto y hasta su finalización. Este tipo de supervisiones, la mayoría de las veces, lo realiza el líder del proyecto con el equipo de aseguramiento de la calidad del software y el encargado de realizar el *testing*. En ellas se revisa la calidad de los documentos que se generen, la funcionalidad del *software* y los niveles de seguridad que se van implementando, lo cual permite llevar un control del personal y de las actividades que se desarrollan dentro del proyecto y que estén alineadas con la metodología que se estableció. Esta actividad es muy importan-

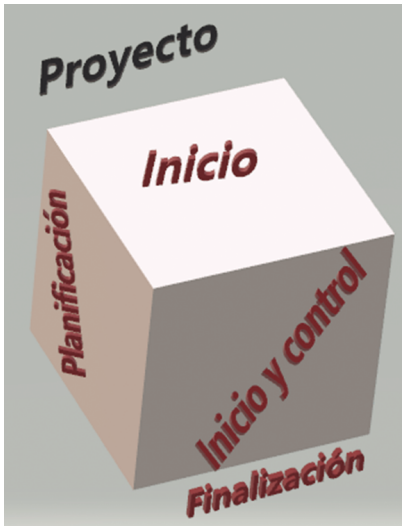

Figura 1.
Administración de proyectos.

Fuente: elaboración propia.

te, ya que ayuda a medir el avance del producto en el ciclo de vida, así como permite realizar acciones para gestionar de forma adecuada la construcción de este a lo largo del desarrollo.

- *Finalización*. En esta etapa se realiza la implementación y la entrega del producto con su respectiva documentación (manual de usuario, manual técnico) y se brinda una capacitación sobre el manejo y las acciones a tener presente para la seguridad del *software*, las cuales son:
  - Instalación e implementación del *software*: se instala y se pone en marcha; esta acción debe realizarse de forma gradual para evitar traumatismo en su operación, ya que en la mayoría de las ocasiones suelen presentarse algunos fallos.
  - Pruebas: estas acciones se realizan en la parte de desarrollo del producto a medida que se cubran los requerimientos establecidos; lo más recomendable es realizar pruebas de manera progresiva con el objetivo de revisar su funcionalidad y las posibles fallas que esté presente, además de que es muy importante tener a la mano el documento de especificación de requerimientos de *software* para determinar si se está cumpliendo con lo establecido en este tanto funcionalmente como con características de *hardware*, seguridad y calidad.
  - Capacitación: según la dimensión *del software*, a pesar de contar con un manual de usuario, es muy importante realizar la capacitación de todos los usuarios del producto dentro de la organización; si es un producto web se debe tener el espacio que contenga videotutoriales. Además, se recomienda realizar conferencias y medios didácticos que sirvan de guía al usuario para comprender y facilitar el uso del *software* (Yourdon, 1993).

Proceso de *software* y métrica de proyectos

Cuando iniciamos un proceso de desarrollo de *software* muchas veces no sabemos por dónde empezar, ya que comprende la realización de una gran cantidad de actividades. Por eso es importante en el momento de contemplar la idea de emprender un proyecto identificar cuál es la metodología apropiada para aplicar en este. En esta disciplina encontramos diferentes modelos, de modo que es importante conocerlas a fondo, ya que cada una de ellas contempla fases y actividades según su pertinencia, como, por ejemplo, el análisis, el diseño, el desarrollo, la implementación y la prueba.

Cada metodología cuenta con sus propias fases y actividades diferentes, pero no basta con esto para que el *software* cumpla con un alto nivel de calidad. Es muy importante adoptar estándares de calidad y métricas, esto es, técnicas que se utilizan para cuantificar y medir la eficacia y la eficiencia del proceso del desarrollo de *software*. Estas fórmulas se aplican y deben revisarse, así como analizarse con el fin de identificar acciones buenas o malas; en efecto, lo que se hace es obtener evidencias y, según estas estimaciones, determinar cuáles

son las mejoras que se deben realizar. Además de esto permite proporcionar una visión profunda del proceso y la comprensión completa y adecuada del proyecto.

Cuando se realiza la evaluación del producto se aplican métricas y se lleva a cabo un análisis exhaustivo, con el objetivo de medir los recursos necesarios, reducir el costo total del producto y establecer adecuadamente el tiempo de duración del desarrollo (Pressman, 2007).

La norma ISO 9000 ISO/IEC 9126 enmarca estos atributos que deben contener las siguientes características:

- funcionalidad: adaptabilidad, exactitud, interoperabilidad y seguridad;
- usabilidad: comprensibilidad, aprendizaje, operabilidad y atractivo;
- mantenimiento: análisis, cambio, estabilidad y prueba;
- confiabilidad: madurez tolerancia a fallos y recuperabilidad;
- eficiencia: comportamiento del tiempo y uso de los recursos;
- portabilidad: adaptabilidad, instalación, coexistencia y remplazo.

Planificación del proyecto

Esta es una de las actividades que requieren un buen esfuerzo y dedicación. Esto en la medida en que a partir de una buena planificación los resultados vendrán de la misma forma, dado que se enfoca en establecer el recurso humano que hará parte del proyecto, además de identificar las responsabilidades de cada miembro del equipo, de manera que se asignarán las actividades a desarrollar en cada fase, el tiempo de duración y el costo por actividad en el que se contemplan los insumos necesarios para el buen desarrollo de la actividad. Para esta fase se recomienda utilizar herramientas tecnológicas con miras a generar el diagrama de actividades, es decir, hacer la programación de cuándo se debe entregar cada una de las obligaciones de cada fase de la metodología de desarrollo de *software* para dar cumplimiento a las metas establecidas, con el objetivo de entregar el producto en el tiempo establecido. Esto permite tener un mejor control de los avances del proyecto, de los recursos financieros y la optimización del rendimiento tecnológico. Sin embargo, es importante establecer las situaciones que se puedan presentar en "efectos positivos y efectos negativos", a fin de tomar las decisiones adecuadas y de forma oportuna, lo que permita evitar retrasos que puedan afectar la ejecución del proyecto. Por esta razón, los objetivos deben ser medibles y reales, de forma que los resultados puedan obtenerse y dar cumplimiento con el objetivo del proyecto. También permite dimensionar un marco estratégico de trabajo en el que el gestor pueda hacer estimaciones lógicas y razonables de los recursos, los costos y la planificación en un lapso apropiado, con lo cual se revise constantemente, ya que debe estar actualizado según el avance del proyecto de *software*. En este sentido, se plantean las siguientes acciones.

1. Identificar el contexto del *software* es la principal tarea, es decir, evaluar el medio ambiente en el que va a funcionar. Se deben especificar restricciones y la capacidad de rendimiento como características propias y el sistema operativo en el que se implementará. Además, se deben establecer reglas claras dirigidas a directivos y técnicos, de modo que se fijen como un prerrequisito para la estimación.
2. El estudio de factibilidad es la segunda a desarrollar, ya que se debe analizar la viabilidad técnica, tecnológica y operativa a fin de conocer el costo aproximado del producto y determinar si es factible desarrollar el proyecto, ya que las organizaciones realizan un análisis profundo y se basan en el costo/beneficio para dar el aval al proyecto.

   Cada recurso queda especificado mediante cuatro características:

   - ⊙ caracterizar el tipo de recurso requerido;
   - ⊙ informes de existencias de los recursos;
   - ⊙ fechas programadas en las que se requieren los recursos;
   - ⊙ tiempo durante el que serán ejecutados los recursos (Sommerville, 2005).

Análisis y gestión del riesgo

El proceso de análisis de riesgo arroja, por lo general, un resultado que se plasma en una matriz de riesgo, en la cual se visualizan todos los componentes reconocidos y la forma en la que se vinculan las operaciones realizadas. La identificación de riesgos depende de dos factores primordiales: la comunicación abierta y asertiva, y la capacidad de prever situaciones embarazosas que se presenten en el futuro. En este propósito es fundamental la participación del equipo para la generación de ideas, la concreción de experiencias y las apreciaciones para identificarlos; es en este punto que la gestión de riesgos nos define una estructura operacional y organizacional mediante la aplicación de procedimientos que nos ayudan a minimizar el nivel de zozobra y evitar que los riesgos pasen a ser problemas, ya que el riesgo, más que una amenaza, representa un desastre. Cada vez que emprendemos un proyecto de *software* y realizamos cambios circunstanciales en él encontramos algunos circunstancias que pueden ser latentes, de manera que se debe realizar un análisis cualitativo y cuantitativo de vulnerabilidad a la que se encuentran expuestas las actividades en desarrollo del producto, así como la posibilidad de que ocurra un evento negativo y no se puedan aplicar las acciones apropiadas dirigidas a acoger, reducir, aceptar y eludir eventos que originen riesgo.

En el 2002 Microsoft definió algunos principios básicos que se deberían tener en cuenta en la gestión de riesgos. En la Tabla 1 se presentan.

Tabla 1

Principios básicos de la gestión del riesgo

Principios básicos	Gestión del riesgo
Ligereza	Requiere que el personal del proyecto continuamente evalúe y gestione dinámicamente los riesgos en cada una de las etapas del proyecto, ya que constantemente se presentaran modificaciones en cada faceta lo que sin duda presentara cambios en los riesgos, en encausamiento con iniciativa facilita que el equipo aproveche los cambios y los convierta en conveniencia.
Aumentar la comunicación	Los riesgos deben de discutirse de forma abierta, con todos los interesados del proyecto, donde participen, en mayor o menor medida de las diferentes tareas relacionadas con la gestión. Los miembros de un equipo no deben tener reservas de ningún tipo para comunicar sus opiniones con libertad y de esta forma evaluar con mayor precisión el estado del proyecto y tomar decisiones consensuadas.
Participación indispensable	Involucrar a todos los miembros del proyecto en la gestión de riesgos estimula la responsabilidad de todos los participantes en el equipo a pesar de que esta se encuentre asignada a la administración.
El riesgo es inseparable de los proyectos	Todo proyecto siempre estará amenazado sea por pocos o muchos riesgos dependiendo del contexto.
Iniciativa	Debe haber iniciativa para el tratamiento de los riesgos, ya que es la forma más eficiente para el tratamiento de estos, lo cual posibilita: 1. Adelantarse a los problemas, para no tener que solucionar después 2. Aplicar acciones en la base del problema y no sus manifestaciones. 3. Disminuir los periodos de solución y reducir el perjuicio producido por una dificultad en situaciones de peligro.
Las circunstancias no deben juzgarse solo por el número de riesgos:	Siempre percibimos los elementos de riesgo como algo negativo, los proyectos no deben calificarse por la cantidad de riesgos localizados. Se debe tener presente que, un riesgo es solo la posibilidad, no la certeza de una perdida ni un resultado por debajo de lo esperado.
Evaluación constante	La gestión y la evaluación de riesgos deben realizarse en todo el ciclo del proyecto, ya que los constantes cambios operativos exigen las valoraciones frecuentes de estado de los riesgos, obligando a actualizar los planes para prevenir o actuar ante ellos. Se debe buscar constantemente la posible aparición de nuevos riesgos.
Aprender de las experiencias	El aprendizaje ayuda a mejorar los resultados, el conocimiento obtenido en un proyecto puede reducir la incertidumbre de la toma de decisiones en otros proyectos cuando la información es poco fiable, el análisis directo de los resultados de los proyectos anteriores fomenta el aprendizaje dentro del equipo mediante el intercambio de opiniones entre sus miembros.
La identificación de riesgos es algo positivo	Para una gestión de riesgos efectiva se deben identificar y determinar a que se enfrenta el equipo del proyecto, este proceso permite al equipo gestionar los riesgos de forma eficaz ya que posibilita la forma de tratarlos, esto implica considerar los riesgos como la única forma de crear la oportunidad adecuada para conseguir los objetivos del proyecto.

(*Continúa*)

Principios básicos	Gestión del riesgo
Especificar y luego administrar	La estructura entre la gestión de riesgos y la toma de decisiones lo constituye la incertidumbre. Cuando un riesgo se define de forma adecuada no deja lugar a dudas y permiten: a) Asegurar la participación de todos permite que el riesgo se trate de la misma forma. b) Comprender las causas de los riesgos y la relación con los problemas que puedan derivarse. c) Disponer de una base para realizar un análisis formal y cuantitativo y planear los esfuerzos asociados a las acciones relacionadas con el proceso de administración del riesgo. d) Aumentar la confianza que los patrocinadores tienen depositada en la capacidad de equipo.

Fuente: Microsoft (2002).

A continuación, se muestran en la tabla 2 algunos ejemplos de riesgos en proyectos de ingeniería del *software*.

Tabla 2

Análisis y gestión del riesgo

Riesgos del proyecto: • incremento en el costo • desbordamiento organizativo	**Riesgos tecnológicos:** • implementar *hardware* nuevo • interactuar con *software* sin probar • componentes del programa radicalmente diferentes a los desarrollados hasta ahora
Riesgos al entorno de desarrollo: • generadores de código apropiados para la aplicación • herramientas de análisis y diseño adecuadas • herramientas de gestión y configuración apropiadas	**Riesgos relacionados con el cliente:** • falta de claridad en las ideas • poca disponibilidad para especificación de requerimientos • no se relaciona con el equipo de desarrollo

Fuente: (Pressman, 2001).

Proyección y seguimiento del proyecto

Una de las formas en las que podemos garantizar el desarrollo adecuado del proyecto es seleccionar una metodología también adecuada, ya que esta nos indica sus fases y sus actividades. Por cada actividad establecemos una fecha de inicio y la fecha de finalización; en esta programación se debe incluir el presupuesto y los insumos necesarios para el desarrollo de las actividades. Un factor que no debe faltar es mencionar quién va a ser el

responsable de emprender la actividad, pues a fin de asignar el recurso humano se menciona el rol del responsable mas no el nombre del profesional, tal como lo muestra la Tabla 3.

Figura 2

Proyección y seguimiento del proyecto

Nombre de la actividad	Fecha de Inicio	Fecha Final	Insumos	Responsable
Levantamiento de requerimientos	5/04/2021	31/04/2021	PC, papelería, videobeam, internet, extensión	Analista
Construcción documento E.R.S	1/05/2021	8/05/2021	PC, Papelería	Auxiliar

Fuente: elaboración propia.

En ocasiones la actividad de estimación del tiempo de duración de una actividad es difícil y más aún cuando el proyecto es grande. Lo recomendado es recurrir a profesionales que tengan experiencia, pues ellos nos ayudarán a realizar la estimación un poco más acertada o, en efecto, podríamos orientarnos por proyectos anteriores en los que se realizaron actividades similares. Una vez tengamos el cálculo del tiempo, de los insumos y el costo, proporcionamos esta información al agregar un 20 % adicional en todos los aspectos. Esto con el objetivo de que si se presentan inconvenientes tengamos un rango de tolerancia para minimizar riesgos.

Cuando la magnitud del proyecto es muy grande se requiere realizar un esfuerzo mayor en lo que concierne a planificación, coordinación y control; si el proyecto es pequeño el esfuerzo en estas actividades es menor, sin embargo, se recomienda realizar todo este tipo de acciones a fin de no dejar nada al azar. Cuando un proyecto es de elevada complejidad el nivel de incertidumbre es más delicado, por lo que se recomienda crear planes detallados y específicos con la información suficiente que permita enfrentar acontecimientos probables dentro de cada fase.

La mejor estrategia para el líder del proceso de planificación consiste en adoptar un planteamiento estructurado, al discriminar cada actividad con la descripción detallada y utilizar instrumentos de trabajo que faciliten la programación y el control del proyecto, como, por ejemplo, los que se enlistan a continuación.

- *EDT o estructura de descomposición del trabajo*. Es un paso inicial para la creación del cronograma de actividades; es importante usarlo con el fin de identificar el trabajo más significativo a desarrollar.
- *Diagrama de Gantt*. Es una herramienta fundamental para el seguimiento y el control del proyecto. En el diagrama se plasma la actividad, la duración y el equipo o la persona responsable, así como se puede visualizar si hay avances; también representa

mediante dos ejes el horizontal (escala de tiempos y el vertical) y sitúa las actividades. Un inconveniente es que no muestra la relación entre las diferentes actividades, lo cual hace difícil determinar el impacto que causa sobre el proyecto el retraso de alguna actividad.

Los grafos de planificación, como, por ejemplo, el método PERT7 CPM, muestran las relaciones entre las actividades utilizando flechas y en el CPM se cumplen cuando la duración de tareas es desconocida.

Una vez que se especifiquen las actividades, su orden de desarrollo y la matriz de compromisos, la cual consta del listado de tareas, subtareas y el recurso humano o unidades organizacionales, se asignarán responsables para cada una de ellas. Solo de esta forma será posible obtener un cronograma adecuado para la planificación, la cual nos determina el inicio y el final de cada una de las labores al estimarlos recursos y cuándo los emplearemos en el propósito de lograr el alcance del proyecto (Pressman, 2001).

Calidad en el *software*

Calidad es la condición por la que un grupo de propiedades relativas satisfacen unas condiciones (Norma ISO 9000.2000). La calidad en un proyecto de *software* se evidencia cuando se da cumplimiento a los requerimientos establecidos en el documento de especificación de requerimientos del *software* (ERS). según estándares internacionales, como, por ejemplo, las normas ISO o IEEE. Para lograr este criterio se requiere establecer fundamentos claros en los que el equipo del proyecto ponga en marcha un plan que permita asegurar la calidad del *software*. Si el proyecto es de gran envergadura, debe crear un equipo internamente que se encargue de este proceso desde el momento en el que se inicie labores, con el objeto de reunir las propiedades necesarias para medir y verificar el nivel de satisfacción de todos los requisitos del sistema de información.

Calidad es también la "ejecución de las operaciones utilitarias y de productividad visiblemente determinadas, de los patrones de desarrollo debidamente ilustrados y de las propiedades comprendidas y esperadas del desarrollo de *software* profesional" (Pressman, 2002).

A fin de realizar un plan de aseguramiento de la calidad del software se debe involucrar las siguientes actividades:

- definir objetivos, metas, procesos y estándares de calidad para el proyecto;
- seleccionar las metodologías, las técnicas, las herramientas y las estrategias a utilizar;
- generar el cronograma de revisiones técnicas formales, las pruebas, las evaluaciones y las auditorías;
- el equipo de aseguramiento de la calidad del software, al igual que en la fase de análisis y diseño, debe elaborar la documentación de las actividades realizadas en este proceso,

coordinar, controlar y gestionar los cambios que se realicen, así como recopilar y analizar métricas de *software* establecidas;

- controlar y verificar cada una de las actividades de la ingeniería del *software* respecto a las adaptaciones que realicen en el proceso de desarrollo del *software*.

Configuración del *software*

Consiste en identificar y gestionar cada uno de los componentes que conforman el sistema y permiten su propia protección y control sobre el progreso del desarrollo del *software* y las funciones que se implementan en este. Su objetivo es establecer y conservar la integridad del producto a lo largo del ciclo de vida con miras a minimizar errores y potencializar su durante vida útil. Estas actividades deben diseñarse eficazmente, de modo que permitan generar un control de cambios que se lleva a cabo en todo el proceso. Es importante identificar mecanismos que permitan administrar cada una de las versiones que resulten y se establezcan relaciones entre ellas, en las cuales se identifiquen los cambios realizados; estas se deben documentar a fin de estar en capacidad de controlar e informar al equipo del proyecto.

El objetivo es mantener la integridad de los productos y establecer el mantenimiento y la preparación de la configuración del *software*. Estas actividades de seguimiento y control deben iniciar con el proyecto y culminar cuando se hace la entrega, teniendo en cuenta que el mantenimiento se realiza una vez se implementa el *software*, esté en funcionamiento y se entregué al cliente.

Para solucionar las necesidades y brindar la gestión de configuración se requiere de las siguientes actividades: planificación, procesos, cultura, productos, automatización, gestión, plan, estrategia y adopción de *supply chain management* (SCM), en español "gestión de la cadena de suministro".

Las primeras siete actividades se refieren al conocimiento del problema y al trabajo que se requiere para plantear la solución; los otros tres son los resultados de estos siete elementos (Pressman, 2007).

Determinación del coste del *software*

3

A fin de estimar el coste o valor del *software* se debe aplicar una variedad de técnicas y procedimientos, con el objetivo de dar a conocer a la organización el valor total antes de iniciar el proyecto, así como que los directivos lleven a cabo su respectiva valoración y determinen si vale la pena asumir la inversión y emprender el proyecto.

Estas técnicas deben involucrar todas las actividades que se realicen en el proyecto, lo que incluye acciones desde el análisis y el diseño, hasta el desarrollo y la implementación del *software*. En este se deben considerar todos los recursos necesarios, en lo que concierne a esfuerzo de recurso humano, tiempo, conocimientos y extensión; es muy importante entender que se debe dar garantía frente al funcionamiento, la calidad, el rendimiento, la seguridad y la eficacia del producto.

También es importante generar un control sobre los costos con un nivel alto de exactitud en todo el proceso de desarrollo, y contar para esto con datos precisos a fin de evitar sobrecostos en materia prima. Lo anterior permitirá mantener de forma adecuada el plan que se generó para la ejecución del proyecto en todas sus aristas (p. ej., costos, recursos y plazo). Este ejercicio de peritaje debe realizarse continuamente con el objeto de enriquecer la actividad, ya que el mercado constantemente sufre variaciones, lo que permitirá, en algunas ocasiones, la reducción de costos o un efecto contrario. Se debe tener presente que siempre es necesario determinar el cálculo de costo/beneficio con el objetivo de optimizar el uso de recursos y alcanzar un muy buen nivel productivo, pues esto se fundamenta en comparar constante el valor del proyecto frente a los recursos utilizados en una medida de tiempo.

Los métodos de estimación nos conducen a obtener una percepción cercana del costo real del proyecto. Esto implica aplicar técnicas y modelos tales como estándares de calidad, técnicas de estimación, métricas de funcionalidad, métricas de producción, métricas dirigidas al tamaño y orientadas a la persona, medidas relacionadas con el tamaño del código, líneas de código (*lines of code* o LOC por sus siglas en inglés), medidas relacionadas con la función (puntos de función o FP por sus siglas en inglés para *function points*), los puntos de objetos (PO), el Cocomo o modelo constructivo de costo (meses/capital humano a aplicar al proyecto), Cocomo II y modelos algorítmicos de costes en la planificación.

Estas acciones permiten viabilizar tareas. Obviamente, como en toda actividad existen riesgos, por tanto, es importante construir modelos y contar con información exacta para estar en capacidad de determinar un cálculo más exacto sobre el valor del *software*, ya que es un factor clave para que el cliente determine si realiza el proyecto o, en efecto, busque

otra solución, pues él espera que el costo del producto sea igual al costo estimado. Por eso es importante hacerlo de forma clara, precisa, real y predictiva, de lo contrario, el proyecto podría fracasar; esta fase es importante para predecir el valor de variables concretas dentro de un proyecto y determinar su viabilidad (Sonmerville, 2005).

Productividad

En el desarrollo de proyectos un tema clave es el nivel de productividad. Este consiste en medir y validar el rendimiento del recurso humano, lo cual permite abstraer una concepción clara sobre la cantidad de personas que van a participar en el proyecto, más el tiempo que deben durar las actividades y, en efecto, determinar el tiempo que se debe contratar cada profesional según su responsabilidad. Estas acciones sirven para fomentar la productividad sin descuidar la calidad. A fin de lograr este objetivo es necesario disponer de herramientas efectivas que conduzcan a aumentar el nivel de productividad y que no sea solo de forma hipotética si no que se materialice para generar una buena sensación en el momento de realizar *benchmarking* (evaluación comparativa) de sus datos.

Para dimensionar el tamaño de un *software* se requiere conocer los puntos de función (FP) encausados a la función, así como las líneas de código orientadas al tamaño. Con estos procedimientos podemos identificar el rendimiento del *software* más el esfuerzo realizado para su desarrollo (Sommerville, 2005).

Técnicas de estimación

Las técnicas de estimación son el soporte de toda planificación al estimar diferentes opciones para calcular el costo total del *software*. Con la finalidad de que este valor se estime lo más preciso posible se debe dimensionar el tamaño mediante el desglose de las actividades, estimar el tiempo y los insumos necesarios. También es importante determinar el entorno. Estas variables nunca serán exactas, lo cual hace que el proceso se torne complejo. Sin embargo, entre más técnicas o herramientas utilicemos para la estimación de costos es posible lograr una aproximación al valor real del proyecto. También se requiere contar con información histórica y de confianza tanto en las métricas como en la experiencia, y para esto se cuenta con algunas técnicas y modelos de estimación como las que se describen a continuación.

- *Estimación análoga*. Esta técnica se concentra en la estimación de costos con base en la comparación de los proyectos registrados en la organización.
- *Estimación juicio experto*. Se apoya en la experiencia profesional, lo que permite una mayor apreciación de los riesgos, los contratiempos, las restricciones y las hipótesis que se enfrentan, así como garantizar estimaciones exactas en las que se ajustan discernimientos de expertos para obtener valoraciones, como, por ejemplo, el método Delphi.

Según Hihn y Habib-Agahi (1990), un estimador experto será preciso si su experiencia es actual y relevante.

- *Estimación por descomposición*. Consiste en desagregar un proyecto en otros más pequeños o en temas de nivel inferior mediante la estimación del esfuerzo empleado *para lograr cada una de ellas*.
- *Estimación paramétrica*. Sirve para determinar el costo de implementación de una aplicación basada en casos confirmados.
- *Estimación de tres puntos*. Es un proceso descriptivo y analítico llamado "Programa de evaluación y revisión técnica" (Pert); se utiliza mediante la armonización de tres estimaciones independientes fundamentada en los contextos pesimistas, optimistas y probables.
- *Estimación por puntos de función y líneas de código como engranaje*. Sirve para diagnosticar la complejidad y el tamaño del *software* a desarrollar.

Tabla 3

Modelo de estimación

Modelo	Característica
Contiene una o varias variables estáticas	Se basa en aplicar funciones y constantes a algunas propiedades del proyecto, como, por ejemplo, LOC (líneas de código).
Cuenta con varias variables dinámicas	Estas miden el tiempo del proyecto frente a su costo mediante una distribución obtenida de manera empírica.
Teóricos con algoritmos prediseñados	Emplean una hipótesis para realizar una predicción a través de una función obtenida según información histórica.

Fuente: elaboración propia.

Según lo indica Kitcheman, las técnicas de estimación aportan procedimientos más asequibles para estos patrones porque disponen el enlace dentro de diferentes variables de costo. Estos procedimientos se denominan "métodos de restricción" y son los que predicen el coste de un procedimiento para modelos de elementos experimentales; entre ellos se encuentran los modelos de restricción de Putnam, Jensen Cocomo o Parr.

Los métodos de componente empírico son esfuerzo de Cocomo, Wolverton, Softcost, Estimacs o Price. El modelo algorítmico Cocomo el más utilizado (meses/hombre a aplicar al proyecto) (Sommerville, 2005).

Modelo de Cocomo

El modelo constructivo de costes (*constructive cost model*) o modelo de Cocomo se fundamenta en la magnitud del producto finalizado. Su primera versión salió a la luz en 1981 (Barry W. Boehm) y es un modelo que consta de tres niveles: simple, moderado

y empotrado. Desde su creación hasta ahora ha sido el más utilizado, ya que emplea el cálculo de la ecuación del trabajo para estimar la cantidad de recurso humano. mensualmente, así como los costos convenientes para el avance, el suministro y el mantenimiento del proyecto. También mide el tamaño de este, basado en líneas de código, principalmente expresadas en SLOC de la siguiente manera: esfuerzo = A * tamaño B, siendo A y B constantes dependientes del modo de desarrollo.

Este contribuye con tres modelos distintos para la estimación que, en particular, proponen una perspectiva de claridad y acercamiento.

El modelo inicia con una primera valoración básica y poco elegante, el intermedio que aumenta la precisión con variables opcionales y, por último, el minucioso que es más complicado y constituye una jerarquía que aporta en *el software*, de modo que logra mejores resultados en las estimaciones; estos tres modelos se dividen, a su vez, en métodos que representan el modelo de proyecto.

- *Modo orgánico*. Conformado por un equipo pequeño de desarrolladores expertos que codifican el *software* en un entorno acostumbrado.
- *Modo semiempotrado*. También conocido como semilibre y aplica el entorno intermedio en medio del orgánico y el rígido, en el que se involucra una combinación de individuos expertos y otros que no los son.
- *Modo rígido*. Se conoce como el modo empotrado que cuenta con limitaciones serias porque domina el vínculo funcional siendo o no procedimental; aquí el inconveniente es especialmente complicado y se fundamenta en la experticia, ya que puede que no exista.

Existen también dos modelos.

- *Modelo intermedio*. Toma el cálculo del esfuerzo en función del tamaño, toma líneas de código (LOC) y un multiplicador (m(x)) e integra conductores de coste (medidas) que posibilitan calcular la distribución de desarrollo del proyecto para involucrarlo en la estimación.
- *Modelo avanzado*. Altera el modelo intermedio teniendo en cuenta el impacto de las variables de coste. Tanto en el modelo intermedio como en este introduce un multiplicador que depende de 15 puntos m(x).

A partir del modelo de Cocomo se creó el modelo Cocomo II que tolera la estimación de coste, tiempo y esfuerzo. Este modelo ofrece un marco de trabajo (*framework*) completo para precisar constantes de productividad a partir de datos tales como el plazo y el esfuerzo de proyectos anteriores que, al combinarse, puede adaptarse a ciclos de vida modernos en los que se contemplan ajustes y otras expansiones pertinentes a las peticiones de los ingenieros de *software*. Cocomo II suministra modelos de estimación minuciosa que se dividen en tres.

- *Método de estructura de aplicaciones*. Adecuado para proyectos de elaboración de interfaces gráficas de usuarios que se adapta en las fases o anteproyectos, en ciclos en espiral.

- *Método de diseño inicial*. Sirve para estimar el costo de un proyecto antes de que se constituya su estructura; se incorpora la investigación de arquitecturas electivas de desarrollo incremental.
- *Modelo post-arquitectura*. Supone una claridad aún más evidente de los examinadores de coste de entradas y se usan después de definida la arquitectura.

Ahora bien, Cocomo II presenta cuatro niveles:

- nivel de construcción de prototipos
- nivel de diseño inicial
- nivel de reutilización
- nivel de post-arquitectura

En la gráfica de la Figura 2 presentamos algunas de las fórmulas más utilizadas.

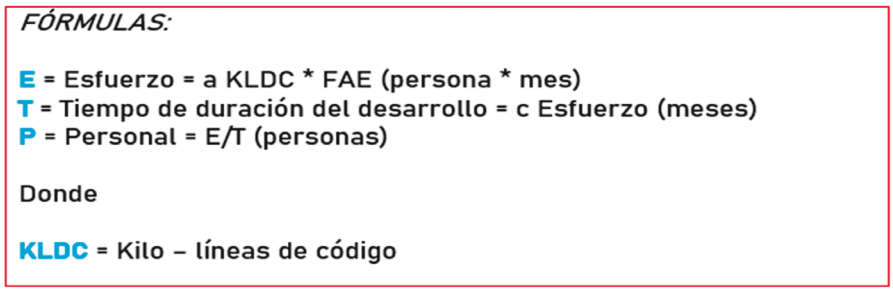

Figura 3.
Gráfica de fórmulas.

Fuente: Sonmerville (2005).

Modelos algorítmicos de costes en la planificación

El modelo de algoritmo de costes se usa especialmente en el desarrollo de *software*. Su objetivo se fundamenta en pronosticar los costes del proyecto con base en la estimación de la magnitud del proyecto, la cantidad de RR. HH. que se requiere y otros insumos que intervienen en la trayectoria del proyecto, ya que acepta realizar análisis cuantitativo empleando la siguiente fórmula: esfuerzo = A^* Tamaño ^{B*} M

Donde:

- A = es un valor constante;
- tamaño = se refiere a la dimensión código del *software*;

- exponente ^{B}: particularmente fluctúa entre 1 y 1,5 M; lo que hace es multiplicar determinadas variables como requerimientos del *software* o la experticia del equipo de desarrollo, al tener presente que esta variable es diferente según la competencia y el conocimiento.

Según Boemh *et al.* (2000), este modelo puede brindar otro tipo de funciones como estimación para investigadores en desarrollo de *software*, evaluar los riesgos y la valoración de la especificación de requerimientos.

Ahora, *B* y *M* son relativas, ya que dependen de la experiencia y el conocimiento del profesional.

Existen tres elementos que se debe tener en cuenta en el coste de un proyecto:

- el costo de equipos (*hardware*);
- el costo de la infraestructura tecnológica (computadores, entornos de desarrollo y más dispositivos) para el desarrollo del *software*;
- el coste de las actividades requerida para la creación del *software*.

La exigencia para estimar el costo de un *software* requiere la utilización de un modelo algorítmico que esté sujeto al volumen de información del sistema. A medida que el desarrollo del proyecto avanza, es más precisa y menos compleja cuando conocemos la clasificación del *software* a desarrollar, y más cuando se ha comparado el modelo usando datos específicos y se ha definido el entorno de desarrollo frente al *hardware* a utilizar.

Es importante tener en cuenta que algunos productos de *software* requieren talento humano y recursos de *software/hardware* especializado (este se debe incluir para consolidar un costo más aproximado a la inversión real. Además, el modelo algorítmico permite estimar riesgos probables del proyecto, y se recomienda aplicar más técnicas, así como comparar los resultados y hacer combinaciones que permitan obtener una mayor precisión.

Figura 4.
Modelo algorítmico.

Modelo	Ventajas	Inconvenientes	Aplicación correcta
Algorítmico	• Entradas y parámetros concretos • Objetividad • Eficiencia en cálculos	• No presta atención a circunstancias excepcionales • Rechazan opiniones subjetivas	Proyectos en escasa alteraciones accidentales, con desarrollo estable y productos sencillos

Fuente: Sommerville (2005).

Duración y personal del proyecto

Una de las actividades importantes en el momento de estimar el costo de un proyecto de desarrollo de *software* consiste en determinar las actividades de cada fase y la cantidad de recurso humano más el perfil del profesional. Esto es una labor coyuntural que muchas veces se complica por la cantidad de actividades y personas que intervienen, como es el caso del analista, el diseñador, el arquitecto de *software*, el programador y demás personal técnico que dará soporte dentro de la organización. La asignación de recursos difiere de asociar a cada actividad el recurso humano, el número adecuado de profesionales según su especialidad, más los insumos materiales que les permitan desarrollar las tareas con un muy buen nivel de calidad.

Para realizar una estimación adecuada se determina la labor a llevar a cabo por parte de cada participante, el tiempo de duración, el valor de la obra de mano por parte del individuo —lo que implica el uso de métricas para evaluar el conocimiento, la especialidad y la experticia de cada profesional con respecto al entorno de desarrollo—, el manejo del lenguaje determinado y las demás herramientas necesarias para la ejecución.

En el desarrollo de un *software* lo que más requiere de asignación de presupuesto es el personal, ya que, generalmente, fluctúa entre un 50 y un 60 %, mientras que el *hardware* o *software* básico e instalaciones consideran el resto del recurso, si se tiene presente que en la estimación se debe incluir el tiempo de dedicación de cada persona participante.

Una vez considerado el esfuerzo requerido para el desarrollo de *software*, es necesario conocer los límites temporales del proyecto en el que se aplican (puntos de fusión-responsabilidades) (Yourdon, 1993).

Ian Somerville (2005, p. 582) afirma lo siguiente: "El modelo Cocomo considera una fórmula para estimar el tiempo de calendario (TDEV) requerido para consumar un proyecto". Esta fórmula es igual para todos los niveles de Cocomo: $TDEV = 3 \text{ X } (PM)^{(0,33+0,2*(B-1,01))}$

PM es el cálculo del esfuerzo y *B* es el exponente calculado (*B* es 1 para nivel inicial de desarrollo de prototipos). Sin embargo, la duración prevista del proyecto y la requerida por el plan de proyecto no son necesariamente la misma. La duración planificada es más corta o más larga que la duración media pronosticada. Sin embargo, existe un límite evidente para los cambios en la durabilidad y el modelo Cocomo II predice esto como: $TDEV = 3 \text{ X } (PM)^{(0,33+0,2*(B-1,01))} \text{ X SCEDPercentage/100}$.

SCDEPercentage es el porcentaje de incremento o decremento en la duración teórica. Si la cantidad prevista demora significativamente la duración estimada, esto indica que existe un riesgo alto de que se presenten inconvenientes y no se pueda entregar el *software* como se programó.

Metodologías de desarrollo de *software*

4

Cuando pretendemos abordar un proyecto de desarrollo de *software*, un factor importante es la aplicabilidad de buenas prácticas. A fin de dar cumplimiento a este apartado es importante seleccionar una metodología apropiada para el proyecto. Pero, ¿qué es una metodología de desarrollo? Es una herramienta que, por medio de una serie de actividades y técnicas permite la administración, la gestión y la ejecución del proyecto de desarrollo de *software*. En cada fase determina una serie de acciones lógicamente estructuradas con el objeto de orientar paso a paso la forma de llevar el proyecto a feliz término, ya que ofrece a los desarrolladores la posibilidad de concentrarse en la calidad del *software* al permitir la detección temprana de errores.

Una de las metodologías tradicionales es el modelo clásico denominado "ciclo de vida del *software*" que consta de seis fases. Cada una de estas presenta una serie de actividades y tareas que se deben abordar en el transcurso del proyecto, que también contempla una serie de técnicas para la gestión y administración del desarrollo. Las primeras fases, al ser ejecutadas, proporcionan un espacio de trabajo para la definición de un proceso descriptivo de desarrollo que contempla acciones orientadas a determinar requerimientos y la estimación de recursos, diseño y mantenimiento, desarrollo de *software*, prueba del sistema e implementación.

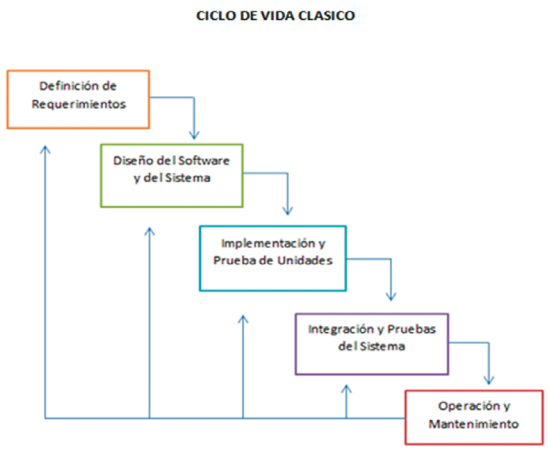

Figura 5.
Ciclo de vida.

Fuente: Pressman (2002).

Estándar para el desarrollo de productos de *software*

Un estándar es un patrón o un modelo desarrollado por expertos cuyo objetivo es proporcionar un arquetipo con una estructura común que sea útil a todo tipo de individuos tales como clientes, distribuidores, programadores, técnicos de mantenimiento, operarios, encargados y personal comprometido en el desarrollo de *software*. Por esta razón utiliza un lenguaje común y circunda en forma de procesos bien definidos que se clasifican en tres tipos:

- Procesos fundamentales: adquisición, suministro, desarrollo, utilización y mantenimiento.
- Procesos de apoyo: registros, gestión de la configuración, garantía de calidad, verificación, validación, revisión unificada, inspección y especificación de problemas.
- Procesos de la organización: gestión, infraestructura, mejora y formación.
- A fin de implementar una metodología o un proceso de ciclo de vida para el *software*, la norma ISO/IEC 12207 considera acciones y procedimientos que se adoptan a partir de la determinación de requerimientos e introduce así la configuración de funciones del sistema hasta la puesta en marcha del producto.

La norma estima que la disposición del estándar se puso en funcionamiento de forma comprensiva para que pueda ser aplicada según los requerimientos de quien le dé uso y se fundamenta en dos principios: el modular, que proporciona la facilidad de ajustar un principio de adhesión y, por otro lado, la responsabilidad, que se asigna a un delegado por cada proceso, de modo que permite el uso del estándar en proyectos en los que intervienen varios individuos o corporaciones involucradas (ISO/IEC, 2008).

Metodologías de desarrollo *de software*

Una metodología para el desarrollo de *software* se compone de varias fases, en las cuales establece unas iniciales para desarrollar en su proceso que permiten administrar el avance del proyecto.

Las metodologías más conocidas se encuentran las que se enlistan y describen a continuación.

- *El modelo clásico o de cascada*. Creado por Winston Royce a fines de la década de los setenta, especifica una estructura de secuencia ordenada por fases; cada una se evalúa en el momento de concluir y con esto es posible determinar si está dispuesta para continuar con la siguiente. Es un modelo muy utilizado y propone una agilidad de desarrollo aceptable, sujeta a cuantiosas críticas, debido a que es limitada e inflexible, lo que complica el desarrollo de *software* actual. A partir de esto el equipo de desarrollo siguen este modelo, pero hacen adaptaciones para que funcione mejor; sin embargo, han aparecido otras metodologías que persiguen desarrollar *software* más velozmente, como, por ejemplo, las siguientes.

- *Espiral*. Se fundamenta en fraccionar el proyecto en miniproyectos que sin querer conducen a aumentar riesgos. Sin embargo, en el desarrollo del proyecto se pueden realizar fusiones con otros modelos, de modo que se puede calificar el de espiral como ambicioso e interactivo, pues se emplean cuatro procedimientos en cada acción del desarrollo: determinar objetivos, analizar riesgos, codificar, comprobar y, finalmente, planificar.
- *Prototipos*. Se utiliza con el objetivo de que el cliente tenga un panorama preliminar del *software* en el que se elabora un prototipo del *software* finalizado determinando los objetivos. Luego se recolecta información para determinar requerimientos y el dominio del bosquejo a fin de hacer una mejor definición. De esta forma el cliente y el usuario comprenden mejor aún como va a ser el resultado del producto cuando los requerimientos sean más convenientes. El prototipo reduce costos e incrementa probabilidad de éxito. Este método puede ser utilizado como actividad para definir requerimientos o tal vez un poco antes como predecesor de estos.
- *Incremental*. Es la fusión de las funciones sobresalientes de la metodología en cascada y la de prototipo. Este proceso de desarrollo siempre incrementa un subconjunto de requerimientos del *software* que no exige una estructura particular para el desarrollo de cualquier otro incremento, ya que está fundamentado para ser funcional y no ser solo una versión previa del *software* final.

También se encuentran otras metodologías como, por ejemplo, la XP o programación extrema, el modelo de desarrollo concurrente, el modelo espiral *win win* (ganar y ganar) o la metodología Scrum, entre otras.

Las metodologías ágiles para desarrollo de *software* son procesos dinámicos cortos que realizan pruebas constantemente y exigen un apoyo insistente de los integrantes del equipo de desarrollo con una planificación cíclica y dirigida. Las metodologías más conocidas son: programación extrema, fuente abierta y métodos de desarrollo dinámicos (DSDM).

Este tipo de modelos requiere que los miembros del equipo sean autónomos, dinámicos, responsables y comprometidos para entregar un producto de muy buena calidad. En este propósito se requiere que cada integrante cuente con buenos conocimientos y la experiencia adecuada que le permita suplir los requerimientos del sistema de informacion, de modo que dé respuesta ágil al proyecto y a la organización. Estos procesos no abandonan la estructura pero reglamentan las ambiguedades en calidad y dinamismo. Estos modelos son actualmente los más conocidos con respecto a métodos ágiles.

Modelos para el desarrollo de *software* web (SIW)

Los modelos para desarrollo de aplicaciones web se fundamentan en una plantilla de producto y un patrón de proceso. La plantilla de producto hace alusión a la abstracción de propiedades o características comunes que existen en cualquier desarrollo; el patrón de

proceso dispone una serie de actividades requeridas para desarrollar el *software* según el patrón de la aplicación.

A través del tiempo han surgido metodologías, métodos y modelos para el desarrollo de *software* web que se fundamentan en estos principios. Algunos de estos se han fusionado según el modelo de proceso y el contexto específico en el que se pueda utilizar.

A continuación, se expone de manera breve cada metodología.

Metodologías para el desarrollo de aplicaciones web

Estos modelos siguen una secuencia de fases con pasos específicos en los que se utilizan métodos de análisis y diseño orientado a objetos y, al igual que en otros métodos, se hace uso de iteraciones.

Los metodos se enfocan en un riguroso proceso en función de la garantía y la calidad de servicio de la secuencia y de calidad en el *software*, de modo que cubren todo el ciclo de desarrollo en el que se emplea una serie de métodos de análisis y diseño con orientación a objetos y se utiliza un tratamiento iterativo.

Los métodos más conocidos son las Metodología para Desarrollar Sistemas de Información Web (Mendoza, 2004).

Modelos para el desarrollo de sitios web descriptivos

Estos modelos se enfocan, principalmente, en el desarrollo de aplicaciones de enseñanza y aprendizaje, en busca siempre de llegar a un resultado de alta calidad. Dan gran importancia al factor humano, a los recursos de aprendizaje basados en web, los recursos de otro tipo y la infraestructura tecnológica para desarrollar el proceso instruccional. Adicional a esto, se procura una reutilización de componentes o recursos para la reducción de tiempos y costos.

Estos modelos tienen como características un rigurozo procedimiento y los componen elementos organizacionales, funcionales, descriptivos y tecnológicos con el objeto de garantizar la calidad del producto para desarrollar sitios web, en los que hacen énfasis en la reutilización de componentes para la reducción de costos.

Algunos ejemplos son el método simple para proyectos de ingeniería (*simple web method*) y el modelado de sistemas descriptivos basados en la web (*systems modeling web-based descriptive*).

Modelos para el desarrollo de aplicaciones de comercio electrónico (e-commerce)

Estos modelos se fundamentan en la reutilización de componentes de *software* y en su combinación para lograr funcionalidades empresariales comerciales que están regidas por

el *habeas data*, ya que este medio se presta para el robo de información y la suplantación de identidad de personas o páginas de fraude. Esto tendrá una serie de características, como, por ejemplo, las firmas certificadas; en Colombia la entidad más conocida para brindar dicha certificación es Certicamar y se da en procedimientos regulados por la Superintendencia de Industria y Comercio (SIC).

En este modelo se reutilizan recursos de *software* y se integran para lograr funcionalidades empresariales para el tipo de aplicaciones en mención.

El más conocido es el marco de referencia basado en componentes para e-*commerce*.

Los ciclos de desarrollo en estos métodos son cortos y se enfocan, principalmente, en el contenido y el diseño de la estructura de navegación de la aplicación. Estos periodos buscan que la aplicación sea entregada en el menor tiempo posible, garantizando su usabilidad, durabilidad y evolución. Al igual que en los modelos para *e-commerce*, procuran la reutilización de componentes dirigidos al desarrollo del producto. Algunos ejemplos son *web site design method* (WSDM) y *object-oriented web-solution* (OOWS).

Estos metodos contienen fases de perfeccionamiento en las que centran su mayor empeño en el planteamiento del boceto de navegabilidad, lo cual permite que se agilice el desarrollo y la aplicación se entregue en el menor tiempo posible asegurando su usabilidad, eficacia y evolución continua según las necesidades de los usuarios. También permiten integrar varias tecnologías como, por ejemplo, la reutilización de componentes y las técnicas de análisis y diseños orientados a objetos.

Modelos para el desarrollo de aplicaciones hipermedia

La mayoría de estos modelos se caracteizan por seguir una secuencia de fases y pasos requeridos para desarrollar SIW, de manera que cubren todo el ciclo de desarrollo para asegurar su calidad.

Estos modelos se enfocan en brindar mayor prioridad a la parte de diseño y no a la de desarrollo. Utilizan dos técnicas de diseño de aplicaciones hipermedia: modelo entidad-relación y técnicas de orientación a objetos. Algunos ejemplos de estos métodos son: modelo de diseño hipermedia orientado a objetos (OOHDM) y metodología de gestión de relaciones (RMM).

Metodología UWE

Esta metodología para aplicaciones web también es un método de autoría que está ligado al planteamiento de ingeniería web orientada a las extensiones UML, las cuales traen las herramientas CASE para diseño de *software*. Las actividades que contempla el modelamiento son el análisis de requisitos, el diseño conceptual, la navegación y la presentación.

Inicia con un análisis de requisitos mediante la técnica de casos de uso, se orienta en la fase de diseño y en el modelo conceptual de la aplicación; se usa como guía para modelar el

espacio de navegación de este modelo. Luego se procede al siguiente paso, un modelo de estructura que muestra cómo navegar en este espacio utilizando elementos de acceso tales como índices, visitas guiadas, consultas y menús. Es decir, el modelo se construye según la estructura de navegación que brinda una UML propicia para avalar bocetos y guiones gráficos. Aparte de esto se sugiere utilizar la interacción de diagramas UML para personalizar el flujo de presentación. En el transcurso de este proceso de desarrollo se determinan acciones que se pueden llevar a cabo de manera automática, adecuando la base para el diseño en aplicaciones web, por medio de una extensión de UML en la que se formulan distintos modelos del proceso de diseño orientado a aplicaciones web.

La metodología presenta un nivel elevado de precisión en las pautas durante los procedimientos de creación de estas aplicaciones. Su fortaleza en el enfoque de ingeniería web se da por el uso exclusivo de notación UML y sus técnicas. Además, cuenta con la especificación de restricciones con OCL (parte de UML), lo que incrementa exactitud en los modelos debido a que permite establecer patrones o principios de coherencia para la construcción de diagrama a diagrama y de diagrama a texto.

El proceso de autoría consta de cuatro eventos que se realizan en el siguiente orden:

- análisis de requisitos (se lleva a cabo a través de modelos de caso de uso);
- diseño conceptual;
- modelo de navegación y presentación;
- modelo de adaptación.

De esta manera, se obtiene como resultados los artefactos que muestra la Figura 7.

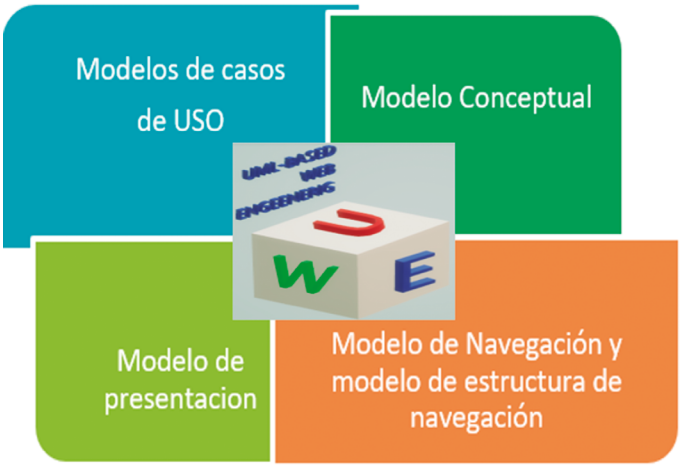

Figura 6.
Proceso de auditoría.

Fuente: elaboración propia.

La clave de UWE se determina como una prolongación acelerada de UML, al proporcionar una figura para el entorno concreto de la web cuya finalidad es suministrar un estilo que permita moldear el ambiente de aplicaciones web que tenga como base fundamental la extensión de UML, ya que cuenta con una serie de procedimientos establecidos por patrones y artefactos que permiten generar croquis metódicos y herramientas de apoyo para la concepción de aplicaciones web.

La UML se apoya en estándares del Object Management Group (OMG), una asociación que se encarga de investigar y establecer estándares con técnicas apropiadas para tecnologías orientadas a objetos, tales como UML, XMI, Corba y BPMN, Model Driven Architecture de OMG, Model Driven architecture (MDA), Object Constraint Language (OCL) y eXtensible Markup Language (XMI). Estas directrices se establecen por medio de manuales para su respectiva divulgación.

Arquetipos que comprende UWE

El procedimiento UWE especifica la creación de seis diagramas que abarcan las fases de análisis y diseño.

UWE establecerá el primer avance para el progreso en el desarrollo del sistema web que introduce modelos de requerimientos al plantear niveles de detalles tales como la eficacia del sistema y describir los casos de uso minuciosamente. Ejemplo de esto son los diagramas de actividad UML, en los cuales se determinan responsabilidades y tareas. Estos casos de uso fueron formulados por el proceso de desarrollo de *software* unificado (RUP) para atraer los requisitos del sistema como técnica que permite establecer los actores en la aplicación.

Existen tres tipos de caso de uso que UWE considera:

- *Caso de uso de navegación*. Se usa para modelar la conducta propia del usuario al utilizar o navegar por medio de la Web App o al requerir datos a través de palabras clave, y se denota <<navigation>> (0).
- *Caso de uso de proceso*. Este se usa para especificar tareas de negocio como actividades transaccionales en la base de datos y no maneja ninguna expresión estándar, así que emplea en este caso una notación innata de UML.
- *Caso de uso personalizado*. Este incide en la personalización del sistema web y se desarrolla por la conducta del usuario denotándose <<personalized>>.

Modelo de contenido o conceptual y modelo de usuario

El diseño conceptual está centrado en un modelo de análisis y comprende objetos implicados en tareas típicas que los usuarios ejecutan con la aplicación.

El modelo de contenido tiene como finalidad brindar una especificación visual de los datos notable para el dominio del sistema web, abarcando, especialmente, el de la aplicación web.

Modelo de navegación

El modelo de estructura de navegación incide en la estructura de nodos y enlaces de una *webapp*, de manera que expone la forma de realizar la navegación y usa componentes de acceso tales como índices, visitas guiadas, consultas y menús. Los elementos de modelado son los siguientes.

Clases de navegación

Una clase de navegación modela una clase cuyas instancias visita el usuario durante la navegación. A las clases de navegación se les otorga el mismo nombre que a las clases conceptuales correspondientes. Para su representación se utiliza el estereotipo UML "navegación clase". Estas clases pueden contener atributos derivados.

Estos atributos se derivan de clases conceptuales que no están incluidas en la navegación modelo. La fórmula para calcular el atributo derivado puede ser dado por una expresión OCL. Un atributo derivado se denota en UML por una barra inclinada (/) antes de su nombre.

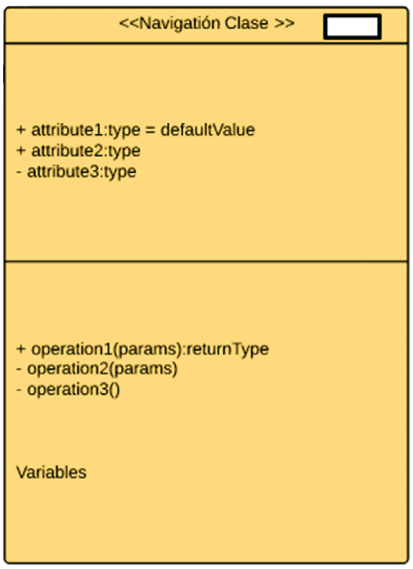

Figura 7.
Clase navegación. Vista de la interfaz de usuario.

Fuente: elaboración propia (realizado en Lucidchart).

Una interfaz de usuario (UI) especifica que cada instancia de esta clase es un contenedor en el que todos los elementos abstractos de la interfaz de usuario se presentan de forma simultánea para el usuario. En las clases de vista de interfaz de usuario se emplea el estereotipo "vista UI".

Modelo de presentación

Es un modelo de estructura e información adicional recopilada durante el análisis de requisitos. El modelo de presentación consta de un conjunto de vistas que muestran el contenido y la estructura de los nodos, es decir, cómo se presenta cada nodo a un usuario y cómo el usuario puede interactuar con ellos. Este diseño de presentación admite la construcción de un modelo basado en la navegación.

Clases de presentación

Una clase de presentación es una unidad estructural que permite particionar una vista de interfaz de usuario en grupos de elementos de la interfaz de usuario. Para las clases de presentación usamos el estereotipo "Clase de presentación" y su icono correspondiente, representado en la imagen vista de interfaz de usuario.

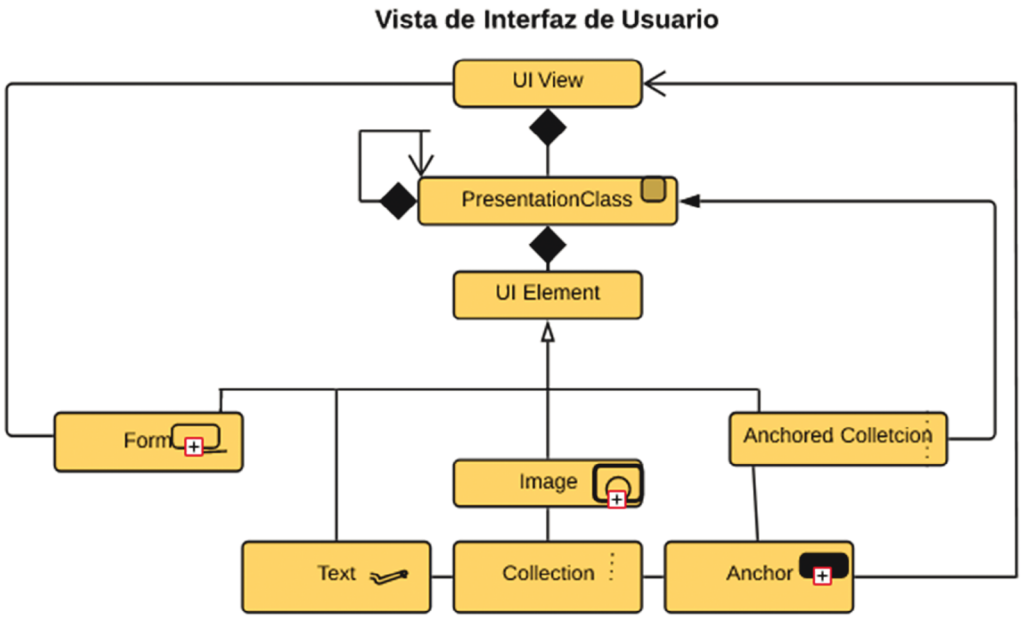

Figura 8.
Vista interfaz de usuario.

Fuente: elaboración propia (realizado en Lucidchart).

Un elemento de interfaz de usuario, por su parte, es una clase abstracta que tiene varias especializaciones que describen elementos de interfaz particulares. Así, por ejemplo, las clases estereotipadas "texto", "imagen", "video", "audio", "ancla" y "forma" son subclases de elementos UI para modelar textos, imágenes, etc. Las clases "Colección" y "Colección anclada" también son subclases de elemento de interfaz de usuario que proporcionan un conveniente representación de compuestos de uso frecuente. El ancla y la forma son los elementos interactivos básicos. Un ancla siempre está asociada con un enlace para la navegación. A través de un formulario, un usuario interactúa con la aplicación web que proporciona información y desencadena un evento de envío (Koch y Kraus, 2020).

Administración de actividades y productos

La administración de actividades y productos es un proceso continuo que se utiliza para gestionar el desarrollo de un producto de *software*, dentro de un plazo previsto y con recursos establecidos. La administración del proyecto involucra no solo a la organización técnica, sino que requiere dirigir recursos humanos y habilidades organizativas, apoyadas por herramientas de trabajo para incrementar la productividad contando con planes estratégicos, tácticos y operacionales y así alcanzar las metas establecidas a través del ciclo de vida del proyecto. Este ha de estar conformado por dos niveles de actividades: las actividades de gestión, que se encargan de la administración, las personas, los procesos y los procedimientos que planifican y construyen el sistema, y las otras actividades que son de desarrollo, centradas en aplicar cada fase de la metodología, agrupadas en áreas funcionales mediante estructura de trabajo para el análisis, el diseño y el desarrollo.

La administración de proyectos se basa en la planificación del proyecto, en una estructura que contiene un conjunto de actividades como, por ejemplo, los elementos organizativos, el proceso administrativo con responsabilidades y la supervisión de participantes, el proceso de desarrollo como métodos, las herramientas, los lenguajes, la documentación, el apoyo y, por último, el proceso con distribución organizada de tiempos en los que se realizan los trabajos.

Los grupos para la administración de actividades deben ser pequeños, máximo de ocho personas. Si el proyecto es muy complejo se debe dividir estos equipos en subsistemas y se define adecuadamente la interfaz y los estándares de calidad, de manera que es el grupo de administración el responsable de la planificación, del desarrollo, la supervisión, el control de tareas y el aseguramiento de cumplir con estándares y tiempo, ejecutando tareas como la generación de propuesta, la estimación de costos, la proyección, el monitoreo, el seguimiento, las revisiones, la selección y la evaluación de personal y la documentación, así como el otro grupo de desarrollo que se encarga del diseño, el desarrollo, las mejoras, el soporte a producción, el mantenimiento, las pruebas y la instalación de paquetes.

Evaluación de modelos y métodos

El modelo de producto es la abstracción de las propiedades más comunes de un producto desarrollado con algún método. De cierta forma este permite la descripción de una aplicación web genérica y la captura de algunos conceptos que permiten definir un modelo de proceso.

Ahora, con el objetivo de conceptuar una aplicación web se diseñó un modelo genérico que representa los conceptos más encontrados en una aplicación web. Este modelo está compuesto por tres capas: presentación, lógica del negocio y datos. La primera describe cómo la información debe ser presentada y está conformada por el navegador y el servidor web. La segunda crea una correspondencia entre los datos almacenados y la información que será presentada al usuario. Por último, la capa de datos se encarga de describir la representación física de estos.

Por lo general, la evalucion de modelo va acompañada de marcos o cuadros de referencias que permiten calificar y posicionar un modelo con respecto a otros.

El marco de referencia usa los principios de la ingenieria de modelos, los cuales establecen que un proceso debe tener especificado un proposito, un dominio, un tipo de problema a solucionar y debe estar conformado por un modelo de producto y un modelo de proceso en el que el modelo de producto especifica los conceptos que manipula el método, así como el modelo de proceso especifica cómo manipularlos.

El marco de referencia esta conformado por cuatro aspectos funcionales: la vista del dominio, la vista del uso, la vista del producto y la vista del proceso. Cada faceta representa un aspecto a evaluar dentro de una vista en la cual está compuesta por un conjunto de atributos. Esta permite clasificar y posicionar un modelo respecto a los demás, con base en una serie de características, como, por ejemplo, su estructura, la calidad de la aplicación web producida o el uso óptimo y acertado de los recursos (humano, económico, tecnológico, etc.), entre otras.

Existe un marco de referencia con una serie de principios y características que debe tener un modelo. Este marco toma en consideración cuatro aspectos: la vista del dominio, la vista de uso, la vista del producto y la vista del proceso. A su vez, existe una serie de facetas que representan un aspecto en particular a evaluar dentro de una vista. Cada vista tiene unos atributos y, a la vez, estos tienen unos valores, con el fin de caracterizar los métodos.

A continuación, se eligen cuatro métodos y se presenta una evaluación según las vistas de referencia. Los métodos son: método basado en componentes para el desarrollo de aplicaciones web; marco de referencia basado en componentes para *e-commerce*; desarrollo de sitios web instruccionales; un enfoque de ingeniería de *software*; y *extreme programming* (XP).

La vista de dominio

Como se mencionó, cada vista está descrita por una faceta, y en este caso lo está por la de alcance, la cual contiene todos los atributos relacionados con el área de aplicación (factores técnicos, gerenciales y/o los que están relacionados con el dominio de la aplicación).

En esta parte de la evaluación se identifica que los métodos basados en componentes para comercio electrónico han sido diseñados con miras a modelos *e-commerce* y sitios web instruccionales. Sin embargo, los métodos para el desarrollo de aplicaciones web y XP) están diseñados para cualquier tipo de aplicación web que se desee desarrollar y, por tanto, podrían ser usados en cualquiera de los dominios.

Ahora, los métodos basados en componentes para el desarrollo de aplicaciones web y el desarrollo de sitios web instruccionales utilizan una orientación de ingeniería de *software* y cubren los aspectos técnicos, gerenciales y relacionados con el dominio de aplicación. Por el contrario de estos, los métodos de escenarios de referencia se basan en elementos propios para comercio electrónico y XP. Asimismo, solo cubren aspectos técnicos, lo cual dificulta el desarrollo del proyecto.

La vista del domio está enfocada en el alcance, la cual, a su vez, se enfoca en el toda el área de domiio de aplicación.

La vista de uso

Está descrita mediante las facetas de aplicabilidad, uso y participación del usuario en el desarrollo de la aplicación web. La primera faceta evalúa las fases que comprenden el ciclo de vida y el enfoque que se le dará a la aplicación. La segunda describe las características del método (visibilidad, estandarización y eficiencia de uso). Por último, la faceta de participación especifica los tipos de usuarios esperados y su nivel de participación en el desarrollo de aplicaciones.

En este caso se evidencia que los cuatro métodos cubren las cuatro etapas de ciclo de vida de una aplicación web y ninguno cubre la etapa de mantenimiento, lo que no garantiza la correcta actualización constante de las aplicaciones.

Se evidencia que cada método tiene un uso en particular: servicios, *e-commerce* o académico (este no posee una orientación en específico).

Ahora bien, todos los métodos muestran qué hacer, pero todos enseñan cómo se tiene que hacer. Esta vista describe por medio de las facetas de aplicabilidad, del uso y de la participacion del usuario en el desarrollo de la aplicación web las características relacionadas con el uso del metodo: visibilidad, estandarizacion y eficencia de uso.

La vista del producto

En esta vista se evalúa el modelo del producto usado por el método y se encuentran dos facetas: la descripción del producto y la descripción conceptual. En esta, el método XP no cuenta con un modelo de producto, en cambio, los métodos basados en componentes para el desarrollo de aplicaciones web, el marco de referencia fundamentado en componentes para *e-commerce* y el desarrollo de sitios web instruccionales (un enfoque de ingeniería de *software*) tienen un modelo explícito que permite caracterizar cada uno de los productos resultantes al utilizar alguno de los métodos en mención.

La vista del producto tiene dos facetas que se seleccionan para describir el modelo del producto: la descripcion del producto y la orientacion y perspectiva.

El modelo de producto que describe a una aplicación web genérica o de prototipo permite capturar los conceptos y los patrones genéricos de cualquier aplicación web en el momento de identificar los criterios que permiten definir un modelo de proceso flexible y ajustable a un contexto particular de desarrollo. Este modelo está organizado en tres capas y presenta la lógica del negocio y los datos. De acuerdo con este modelo de producto se dice que el desarrollo de una aplicación web involucra la especificaión, el diseño, la implementacion y la prueba de tres grupos de componentes distribuidos en cada uno de las capas: capa de presentacion, capa de lógica del negocio y capa de datos.

La vista del proceso

Esta contiene un conjunto de facetas que describen el proceso de desarrollo de la aplicación que se desea desarrollar. Se consideran para este caso cinco facetas:

- modelo de proceso de desarrollo;
- características del proceso;
- gerencia de proyecto;
- cobertura del ciclo de desarrollo;
- proceso de posdesarrollo.

Según la evaluación, todos los métodos aquí siguen un enfoque evolutivo, es decir, permiten hacer cambios a requerimientos dados o de diseño a medida que se evalúa la aplicación con el cliente.

Respecto a la faceta de uso, la característica del proceso y la cobertura del ciclo de desarrollo, no dividen sus fases en pasos y las actividades no se indican de manera clara y precisa. Estos no tienen en cuenta la gerencia de proyecto, es decir, no incluyen actividades relacionadas con la planificación y el control u organización de equipos de trabajo. Solo se tienen en cuenta las revisiones técnicas de validación y verificación.

Acerca de la faceta de cobertura del ciclo de desarrollo, el método de cobertura del ciclo de desarrollo cubre de forma ligera cada etapa que la compone, aunque solo establece una definición informal de requerimientos.

Acerca del método características del proceso, este considera la etapa de análisis y definición de requerimientos. Además, cubre la etapa de pruebas de cada componente (tanto las de atención como las de integración).

Por último, los métodos modelo de proceso de desarrollo y la gerencia de proyecto cubren totalmente las etapas del ciclo de desarrollo.

Esta vista contiene un conjunto de facetas y atributos que describen el proceso de desarrollo de la aplicación que desea producir y se evalúa por el método de cinco facetas: orientación, enfoque, claridad, estructura del proceso, método de ejecución del proceso (Montilva y Barrios, 2004).

Análisis y determinación de requerimientos

5

Un requerimiento es una propiedad o una restricción que debe tener el *software*. Según la IEEE, un requerimiento es la capacidad que deben tener un sistema o los elementos de sistemas para cumplir con una estipulación, un modelo, una distinción u otro documento explícito.

Consiste en recopilar, seleccionar, analizar y verificar la información necesaria e identificar las necesidades del usuario, del cliente y del sistema, así como en que estas se puedan satisfacer. En el proceso de desarrollo de *software* esta actividad es de gran importancia, ya que su finalidad es entender, identificar y especificar los requerimientos del sistema, por tanto, se debe proceder con un procedimiento técnico en el que se involucran muchas personas. Dependiendo del tamaño del *software*, en muchas ocasiones esta actividad se vuelve compleja.

Estas acciones se fundamentan en la recopilación de información (recolección, análisis, negociación, especificación y validación de los requisitos del *software*) y establece una actividad de administración y documentación de requerimientos en los que se involucra la gestión de cambios, el mantenimiento y el seguimiento de estos (Senn, 1997).

Determinación de requerimientos

Quizás es la actividad más importante en el inicio de un proyecto de desarrollo de *software* porque se convierte en la línea base, ya que determina de manera específica las funciones, las características y el comportamiento que debe tener el *software*. Por esta razón se debe estructurar de forma clara y específica para minimizar los inconvenientes relacionados con su desarrollo y, por ende, se deben abordar las siguientes acciones.

1. Analizar el entorno donde se va a ejecutar el *software*, identificar las necesidades del cliente y de los usuarios, saber cuáles son los objetivos y las metas que se pretenden alcanzar con el desarrollo del producto, es decir, conocer el problema para saber cuáles son las expectativas esperadas que genera el *software*.
2. Recolectar información relevante que permita conocer la organización, las normas, las políticas y los estándares, determinar la forma en la que va a fluir la información, conocer los procedimientos y módulos y conocer el dominio que tendrá el *software* en el contexto de negocio, así como la parte legal a nivel nacional e internacional sin afectar los procesos que se desarrollan en la entidad.

3. Determinar restricciones y límites del producto, es decir, determinar qué hace el *software* y qué no hace, determinar claramente el entorno funcional y su medio de acción en torno a la funcionalidad, la seguridad, la integridad, la disponibilidad, la fiabilidad y la usabilidad, esto es, determinar los requerimientos funcionales, no funcionales y demás requerimientos que requiera el *software* cumpliendo estándares de calidad. Los desarrolladores están facultados a seleccionar inicialmente un bosquejo genérico del sistema, con el objetivo de definir funciones básicas y determinar el tamaño del *software*; deben identificar a los usuarios del producto, que son quienes pueden ayudar a determinar los requerimientos y funciones del *software* (Senn, 1997).

Requerimientos básicos

Estos requerimientos nos brindan una concepción del proceso básico y su funcionalidad, lo que nos lleva a centrarnos en la información que nos brinden los usuarios a través de observación directa, encuestas o entrevistas. Estas técnicas nos permitirán obtener información minuciosa y los datos que utiliza, procesa y produce este proceso. Es muy importante identificar el paso a paso de cada operación que realiza el usuario en el sistema, con el objetivo de evidenciar el tiempo que dura en ejecutarse cada evento y la frecuencia con la que fluyen los datos. También se debe identificar quiénes utilizan esta información y qué hacen con ella, a fin de describir funciones, transacciones, procedimientos y la información resultante. El líder del proceso de análisis, una vez que conozca la información proporcionada por el cliente, debe documentar los requerimientos básicos y generar un arquetipo que permita darle solidez a la fase de determinación de requerimientos. En el propósito de estructurar mejor su investigación se recomienda tener presente, según estos aspectos, los siguientes puntos.

- *Entradas al software*. Se identifican los datos de entrada al sistema, el contenido de estos y los detalles funcionales del dispositivo de entrada.
- *Salidas del sistema*. Se determinan los dispositivos de salida (audio, video e impresión) y su función.
- *Funciones del software*. Se deben conocer los datos que ingresan al sistema, los procedimientos y las operaciones que se realizan con estos. También se debe explicar las instrucciones que requiere cada operación del *software* para su buen funcionamiento.
- *Características del software*. Es muy importante explicar los requerimientos funcionales y no funcionales que debe tener el *software*. Esto permitirá establecer su buen funcionamiento y determinar el propósito por el cual se construye. El producto debe ser intuitivo, de fácil uso y amigable, debe permitir que se realice mantenimiento y que se pueda actualizar de forma sencilla; la información debe estar disponible cuando se necesite, y cuando se presenten fallas el *software* se debe recuperar fácilmente.
- *Atributos del entorno del software*. Presenta otros requerimientos adicionales tales como son limitación de los procedimientos, compatibilidad, capacidad máxima de carga permitida y temperatura máxima admitida. A fin de lograr un producto de muy buena

calidad es necesario analizar y documentar de manera muy específica los requerimientos del *software* según sus características y garantizar a desarrolladores y usuarios la comprensión y el uso fácil de la aplicación.

Un *software* de calidad debe cumplir con las características que se muestran en la tabla 5.

Tabla 4

Software de calidad

Requisitos	Definición
Entendible	Se caracteriza porque es fácil de leer y entender, su redacción debe ser simple y clara.
Necesario	Representa características físicas o de calidad que no pueden ser reemplazados por otras capacidades del producto o del proceso.
Consistente	Que no es contradictorio con otro requerimiento.
No ambiguo	Que es claro, preciso, objetivo y entendible.
Completo	No necesita ampliar detalles en su redacción y proporciona toda la información suficiente para su comprensión.
Verificable	Fácil de valorar aplicando métodos de verificación cuantificables (como inspección, análisis, demostración o pruebas), para garantizar que cumple sus propósitos.
Reales	Que pueda ser realizado, que sea factible su implementación en el sistema, teniendo en cuenta que si no existe la tecnología, el personal, la experiencia o el presupuesto para implementarlo no serán factibles.
Correcto	Describe con claridad lo que el cliente o usuario quieren, se recomienda que los revise el cliente y el desarrollador, para asegurar que no tenga errores.
Rastreables	Los requerimientos pueden ser rastreables así: hacia atrás significa que para cada requerimiento se conoce su origen y hacia delante, significa que todo requerimiento este escrito de tal forma que facilite la referencia a los requerimientos con los que se relaciona.

Fuente: Senn (1997).

Requerimientos de las transacciones de los usuarios

Las transacciones que realizan los usuarios a través del *software* exigen requerimientos que capturen, procesen y almacenen datos. Además, con estos estructurar información que requiera el usuario, por lo cual es importante que el proyecto de *software* considere este tipo de requerimientos en los que se recolecten y se categoricen en grupos para llegar a un acuerdo y definir el alcance del proyecto, ya que estos facilitan la navegación en el *software* y así estar en capacidad de utilizar de forma correcta y definir claramente sus funciones. Esto lleva a resaltar la importancia de conocer la forma en la que se llevan a cabo las transacciones, con el objetivo de determinar las transacciones u operaciones en las que interviene el usuario con el *software*. A fin de diseñar la interfaz gráfica de usuario (GUI), según las especificaciones

establecidas (iconos, ventanas, listas de chequeo, cuadros de diálogo, menús, informes, etc.), este tipo de acciones permitirá generar un diagrama de árbol en el que se represente la jerarquía de los nodos y la forma de interacción de todos los componentes a utilizar en las interfaces, con el objetivo de facilitar la navegación del usuario. Una vez que se determinen todos los componentes que se requieren para cualquier transacción y operación (puede ser de cancelación o cierre de la ventana, etc.) deben ser aprobadas por el usuario.

El desarrollo de esta actividad requiere que el analista se reúna con los usuarios intervinientes y se determinen de forma clara todos componentes que debe llevar cada una de las interfaces. En este propósito se recomienda resolver los siguientes cuestionamientos:

- ¿Cómo inician las transacciones?
- ¿Cuáles son los datos de entrada, proceso, salida y almacenamiento?
- ¿Qué usuarios pueden realizar las transacciones?
- ¿Cómo se inician las transacciones y cuál es su propósito?
- ¿Cada cuánto ocurren?
- ¿Qué volumen de datos y qué tipo de datos utiliza la transacción?
- ¿Cuáles son los usuarios clave?
- ¿Qué tipo de restricciones se usan y qué métodos aplica? (Senn, 1997)

Requerimientos de decisión de los usuarios

En un *software* es muy importante contemplar los requerimientos de los usuarios, ya que ellos realizan diferentes operaciones constantemente. En realidad, son ellos quienes deben ayudar a determinar las funciones del sistema, así como a evaluar si es intuitivo o no, ya que el *software* debe responder a estas características, además de que se realicen transacciones de forma fácil.

Así, deben capturar, procesar y almacenar los datos de forma segura, evitando su pérdida, y que en efecto faciliten la toma de decisiones. Sobre todo, porque estos se fundamentan en la información generada por el *software*, la información resultante de estos procesos y la que arroje el sistema, lo que debe permitir a los administrativos tomar decisiones claras y que otorguen beneficios a las organizaciones.

Ahora bien, las operaciones concernientes a las decisiones no contemplan procedimientos estandarizados específicos de acciones. Exigen, en cambio, que los sistemas capturen la información necesaria y apropiada para que sirva como soporte cuando se requiera tomar una decisión. Es muy importante entonces establecer datos precisos y pertinentes que faciliten la realización de operaciones orientadas a realizar las transacciones que arrojen resultados adecuados. Además, que quien tome estos datos para realizar estas acciones sienta que los resultados arrojados por el *software* son 100 % confiables y seguros.

En razón a lo anterior se debe involucrar a los usuarios del sistema para minimizar el nivel de incertidumbre que este pueda generar, ya que con sus aportes es posible encontrar casos inusuales, y es en estos casos en los que se valida la trascendencia del uso de técnicas para la recolección de información tales como cuestionarios, entrevistas, lluvia de ideas, revisión de registros y la observación directa, entre otras que faciliten la solución de estos inconvenientes. Otra forma es acudir a los manuales de funciones de usuarios de la dependencia, las políticas institucionales y la estandarización de procedimientos operacionales que utilizan los empleados y que sirven de guía a los gerentes. Estos documentos y técnicas pueden facilitar el trabajo del analista en el propósito de analizar y comprender mejor las funciones que debe realizar el sistema, así como determinar qué transacciones requieren apoyo con las acciones lógicas dentro de la organización. En este caso la observación directa es la herramienta adecuada porque brinda datos reales y consecuentes sobre la forma en la que se llevan a cabo las actividades e indica como se puede evaluar los procedimientos, razón por la cual los analistas deben conseguir información relevante para resolver cuestiones como las siguientes:

- ¿Cuál es la información útil para la toma de decisiones?
- ¿Cuál es la fuente que provee mayor cantidad de información?
- ¿Qué otros datos son necesarios para realizar las transacciones, pero el sistema no los captura?
- ¿Cuál es la forma en la que se deben procesar los datos para que produzcan información y que apoyen la toma de decisiones?
- ¿Qué tipo de información proveen las fuentes externas? (Senn, 1997)

Requerimientos de la organización

El proceso de desarrollo de *software* implica aplicar el proceso de ingeniería de requerimientos. Estas acciones son lideradas por el analista, quien debe utilizar diferentes técnicas, herramientas y estrategias con el objetivo de obtener información consecuente para clasificarla, filtrarla y, en efecto, analizarla para identificar procesos, subprocesos y la forma en la que se realizan los procedimientos en la organización. Estas acciones son pertinentes ya que permiten establecer los requerimientos del sistema que hacen posible el desarrollo de un producto que brinde funciones convenientes, fluidas y adecuadas. Se recomienda identificar a cada uno de los usuarios que intervienen en el proceso con el objetivo de obtener información veraz y verificable con miras a analizar los datos. Es también muy importante que exista una comunicación asertiva y continua entre el TEAM de desarrollo, los clientes y los usuarios, utilizando técnicas que faciliten la recolección de información adecuada y que permita identificar los requerimientos.

Los requerimientos establecidos deben ser consecuentes con el plan estratégico de la organización, así como las fuentes de las que se obtiene la información deben ser fidedignas,

ya que de esta dependen las funciones y el ámbito del *software*. Los involucrados se clasifican según su rol en el sistema:

- *Usuario final*. Son usuarios internos que usan el *software* y determinan si la aplicación es amigable, el nivel de usabilidad, la disponibilidad y la seguridad del sistema; se vinculan con procesos directos y específicos que realiza el *software* dentro de normas de su ambiente laboral y serán los que utilizan las interfaces y los manuales de usuarios.
- *Usuario administrativo*. Son personajes que comprenden el dominio del *software* y entienden el proceso por el que se implementará; pueden proporcionar información adecuada sobre la interfaz del producto.
- *Analistas y programadores*. Son los encargados de codificar el producto a través del entorno de desarrollo y se encargan de establecer la comunicación directa con el cliente.
- *Personal de pruebas*. Se encarga de realizar el *testing* y efectuar el programa de pruebas para confirmar que las restricciones del sistema determinando son adecuadas; también valida que los requerimientos cumplen las exigencias de los interesados.

Según el Project Management Institute (PMI) los requerimientos de un proyecto pueden dividirse en dos categorías: requerimientos de negocio y requerimientos técnicos. Los primeros definen las necesidades y los deseos de la organización en relación con la consecución del proyecto, mientras que los segundos se centran en las soluciones que harán posible la obtención de dichas metas. Todos son igual de importantes de satisfacer e imprescindibles para finalizar el proyecto con éxito.

Este tema es delicado y se recomienda ser intuitivo, ya que se debe determinar elementos claros direccionados a suplir las siguientes necesidades:

- permitir y procesar los datos para producir la información necesaria;
- la forma en la que debe presentarse la información;
- qué datos se originan en fuentes externas de la organización;
- las rutinas deben ser claras para controles precisos;
- información apropiada para direccionar la toma de decisiones;
- determinar la fuente de información (Senn, 1997).

Técnicas para recolectar datos

Consiste en aplicar una serie de técnicas que permite la recolección de datos por parte de los usuarios y los interesados en el *software*. Con estas herramientas se capturan los datos, se analizan y se obtienen los requerimientos del sistema. Las herramientas más comunes para alcanzar este propósito son cuestionarios, entrevistas, lluvias de ideas y observación directa. A continuación, se describe cada una de estas.

- *El cuestionario*. Consiste en generar una serie de preguntas orientadas a obtener información por parte de los interesados y a la obtención de información pertinente con el fin de determinar los requerimientos del *software* a desarrollar. Estas preguntas se plasman en un formato y se reúne un grupo de personas que participarán de forma ágil. Se puede formular preguntas estructuradas y coherentes a los usuarios del sistema actual o al propuesto, quienes deben proporcionar información consecuente y estar implicados en los procedimientos concernientes al *software* a desarrollar; algunos son administrativos o empleados y proporcionan datos útiles para el sistema.
- *La entrevista*. Esta técnica se usa para obtener información de forma verbal por parte del usuario. El analista debe ser cauteloso y generar un diálogo tranquilo para formular así preguntas que permitan obtener del usuario respuestas concretas, claras, precisas y abstractas sobre cuáles son los datos que se requiere ingresen al *software*, el proceso que se realizaría con ellos y qué salidas se podrían establecer. Esto con la ventaja de que esta información se pueda ampliar para obtener una mayor cantidad de datos. La actividad debe realizase en el inicio del proyecto a fin de obtener la cantidad de datos que permita brindar mejores soluciones.

  Se recomienda que el entrevistador tenga un buen nivel de conocimiento al respecto, experiencia y cuente con habilidades que le permitan orientar una conversación tranquila. De esta manera, podrá indagar a profundidad al entrevistado para alcanzar los objetivos de la técnica. Esto se recomienda ya que suelen presentarse situaciones complejas, pues el procedimiento debe obtener buenos resultados. La información se debe documentar para, posteriormente, analizar los datos que ayuden a determinar los requerimientos del sistema.
- *Lluvia de ideas*. Esta técnica exige realizar reuniones presencialmente o en línea, cuyo objetivo es que cada integrante presente una idea que oriente a la solución del problema. Esta herramienta permite incrementar el nivel de creatividad por parte de los participantes a fin de que surja un mayor número de ideas y obtener un resultado con un alto nivel de calidad. Se debe realizar un filtro con el fin de seleccionar las mejores ideas y, posteriormente, hacer un análisis detallado que permita determinar los requerimientos del *software*. El líder del proceso debe conseguir que todos los interesados participen, además, debe hacer las veces de moderador para que dé inicio a la sesión, conceda la palabra a cada uno de los participantes de forma ordenada y, por último, cierre la actividad. Bernd y Allen (2002, p. 84) afirman:

> Una vez que el cliente y el equipo de desarrollo coincidan en una idea en común, define las formas y los límites del *software* poniéndose de acuerdo en coordinar técnicas, actividades y estándares, desde el inicio del desarrollo hasta lograr la consolidación del *software*.

En la recolección de información realizada con el fin de determinar los requerimientos del *software* se puede aplicar cualquier técnica o todas, de modo que es posible utilizar

preguntas abiertas de las que surjan ideas, se expresen experiencias generales y el paso a paso de la forma en la que se realizan los procedimientos. Este tipo de preguntas permite identificar cómo el usuario percibe e identifica características, funciones, procedimientos y actividades del sistema, etc. Ejemplos de algunas de las preguntas abiertas que se les puede formular a los usuarios son:

- ¿Qué tan importante es resolver el problema?
- ¿Por qué se debe resolver el problema?
- ¿Qué ocurre si no se soluciona el problema
- ¿Cuál es la solución que recomienda el usuario?
- ¿Quién es el usuario?
- ¿Cuáles son sus necesidades?
- ¿Qué espera el usuario frente a usabilidad, confiabilidad y rendimiento?
- ¿Qué tecnología se tiene y cuál se requiere para la implementación del *software*?
- ¿Cuánto tiempo lleva realizando el procedimiento?
- ¿Cuáles son sus habilidades, capacidades y experiencia?
- ¿Cuál es el valor de una solución exitosa?
- ¿Qué perjuicios trae el desarrollo del producto?
- ¿Qué problemas podría causar este producto en el negocio?
- ¿En qué ambiente tecnológico se usará el producto?

- *Observación directa*. Es una técnica que permite al analista ubicarse directamente en el área de trabajo para visualizar la forma en la que se ejecutan los procedimientos realizados en ese espacio de trabajo. Se debe analizar la forma y la duración de cada una de las acciones que ocurren en este, así como analizar muy bien la forma en como se hace, cuántas veces se hace, quién lo hace y la frecuencia con la que se realiza. Si la técnica se desarrolla de forma adecuada, permite identificar, clasificar y documentar eventos reales. Para realizar esta actividad adecuadamente se recomienda seguir estos pasos:
  - establecer los procedimientos a observar;
  - identificar los objetivos por los cuales se va a hacer la observación;
  - identificar las herramientas de consignar la información obtenida.
  - realizar la observación cuidadosamente, de forma detallada y crítica;
  - utilizar herramientas adecuadas que faciliten la documentación de los datos obtenidos;
  - hacer un buen análisis para interpretar la información de la mejor forma;
  - realizar las conclusiones claras, precisas y pertinentes.

Modelos de sistemas

Un modelo es una representación gráfica de conceptos que se hace por medio de diagramas. Sirve para presentar los eventos o acontecimientos más significativos, con el objeto de predecir acontecimientos o sucesos que se han observado y, a través de este, plantear soluciones a problemas formulados.

En el desarrollo de *software* se aplican técnicas dirigidas a documentar la determinación del sistema, utilizando un conglomerado de modelos del sistema. En este propósito conceptualizamos el modelo como una idealización que está en proceso de investigación, en lugar de una representación disyuntiva del sistema, de modo que estos diferentes modelos se grafican para describir cómo interactúan los procesos de la organización, el enigma por determinar y el *software* a realizar. Esta graficación permite que exista una mejor comprensión, de manera que la fisonomía de este es más relevante en la representación del sistema a fin de suprimir pormenores, ya que constituye un enlace entre el proceso de análisis y el de diseño. Sin embargo, se considera que el gráfico de un sistema obligatoriamente debería contener toda la información necesaria sobre la entidad que se está representando, mientras expresa requerimientos del sistema técnicamente para que las personas que no son expertas lo comprendan fácil y rápido gracias a las representaciones gráficas usadas. Por esta razón se utilizan modelos en el proceso de análisis desde diferentes ópticas, tales como la óptica externa en la que se nos da a conocer el entorno del sistema, la óptica de comportamiento que nos muestra obviamente el comportamiento del sistema y, por último, la óptica estructural, en la cual se modela la arquitectura de los datos del sistema.

Sommerville (2005, p. 154) señala lo siguiente:

> 1. Un modelo de flujo de datos: muestra cómo se procesan los datos del sistema en diferentes etapas.
>
> 2. Un modelo de composición o de agregación: muestra cómo las entidades del sistema están compuestas por otras entidades.
>
> 3. Un modelo arquitectónico: muestra los principales subsistemas que componen un sistema.
>
> 4. Un modelo de clasificación: los diagramas de clase/herencia de objetos, muestra cómo las entidades tienen características comunes.
>
> 5. Un modelo de estímulo-respuesta: también llamado diagrama de transición de estados muestra cómo reacciona el sistema a eventos internos y externos.

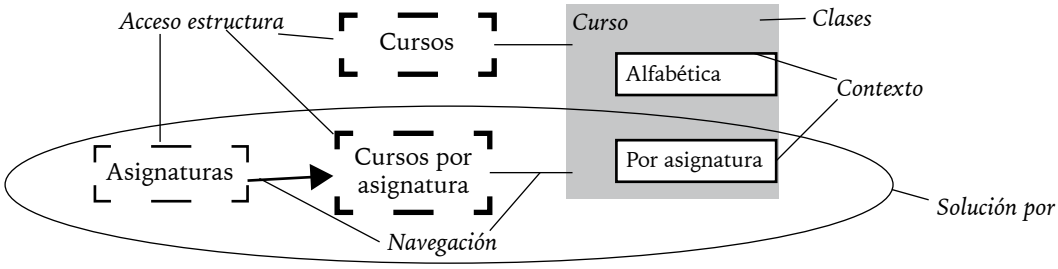

Figura 9.
Modelos de contexto.

Fuente: elaboración propia.

Todos los sistemas que se desarrollan son en realidad subsistemas de uno mayor. Por tanto, es importante definir, primero, las interfaces entre nuestro sistema y el resto del universo, es decir, el ambiente, tal como lo muestra la ilustración de la Figura 8.

Para definir el ambiente utilizamos, valga la redundancia, el modelo ambiental, con el cual es posible modelar el exterior del sistema y su interior (modelo de comportamiento). Así delimitamos las fronteras entre el sistema y el ambiente y sabremos también qué información entra al sistema desde el ambiente exterior y cuál se produce como salida del sistema.

El diagrama de contexto sirve para demostrar el ambiente en el que actúa el sistema, el medio en el que se encuentra y dónde reconoce GUI. Estas son las que facilitan la interacción con otros sistemas en los que se utilizan flujos de control de datos para mostrar las interacciones existentes entre los agentes externos y el sistema mediante la descripción de la información entrante o saliente del sistema a través de las diferentes interfaces y de materiales. Estos, a su vez, permiten distinguir lo que es en sí el sistema y su entorno y qué no, pues implican aspectos sociales y organizacionales del entorno, pero muestran de manera amplia las relaciones entre el sistema que está desarrollando y el medio, es decir, no describe en ningún momento la estructura del sistema de información.

¿Cómo elaborar diagramas de contexto?

Se elabora una burbuja en el centro en la que se ubica el sistema que se va a construir y se rodea de las entidades (como los usuarios del sistema), de otros sistemas con los que interactúa, los reservorios de datos y las bases de datos, a los cuales se necesita acceder para cumplir su función correctamente utilizando terminadores, flujos de datos, flujos de

control y almacenes de datos. Los terminadores se grafican por medio de rectángulos y el flujo de datos o de control se representa por medio de flechas o a través de reservorios externos. Entre los terminadores no debe haber comunicación.

En la ilustracion Herramientas para elaborar un diagrama de contexto se evidencian las herramientas que utilizamos para crear un diagrama de contexto (Yourdon, 1993).

Modelos de comportamiento

La utilidad del modelo de comportamiento consiste en que se explica detalladamente la forma como procede el sistema en su totalidad. Dentro de este se encuentran dos tipos de modelos: los modelos de flujo de datos y los modelos de máquinas de estado. Cada uno nos muestra el procesamiento y la fluidez de los datos y cómo el sistema reacciona a posibles eventos. Estos dos modelos pueden trabajarse unidos o bien sea separados, dependiendo del tipo de sistema que se desarrolle.

Ahora bien, los modelos de flujos de datos o modelo de procesos se definen como gráficos sintetizados de un procesamiento que simbolizan desde un panorama detallado el *software*.

Por su esencia, los patrones son resumidos, es decir, un modelo de procesos es una abstracción de un sistema verdadero. Para la elaboración de estos modelos contamos con varias herramientas *on line* tales como Creately, Visual Paradimg, Plantuml, Yuml, Nomnoml, Textuml y Lucidchart, entre otras.

Inicialmente, seleccionamos la paleta de herramientas según muestra la Figura 9, en la que cada archivo se presenta inicialmente como una pantalla vacía con una paleta de herramientas.

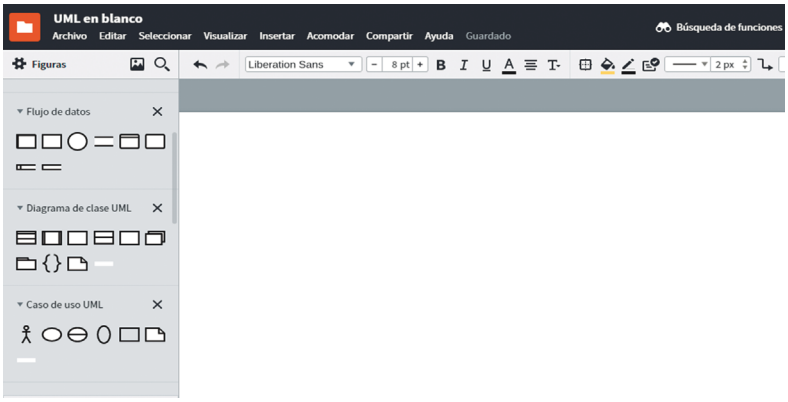

Figura 10.
Paleta de herramientas de Lucidchard.

Fuente: elaboración propia.

Estas herramientas primordiales son el proceso, el almacén de datos, el flujo de datos y la entidad externa. El proceso es, en este caso, un círculo.

A continuación, se usan los mismos estilos tipográficos (proceso, almacén, flujo) y, además, los ítems de datos y las variables se imprimen en fuente Courier new, tal como se muestra en la ilustración herramientas para elaborar diagramas de comportamiento.

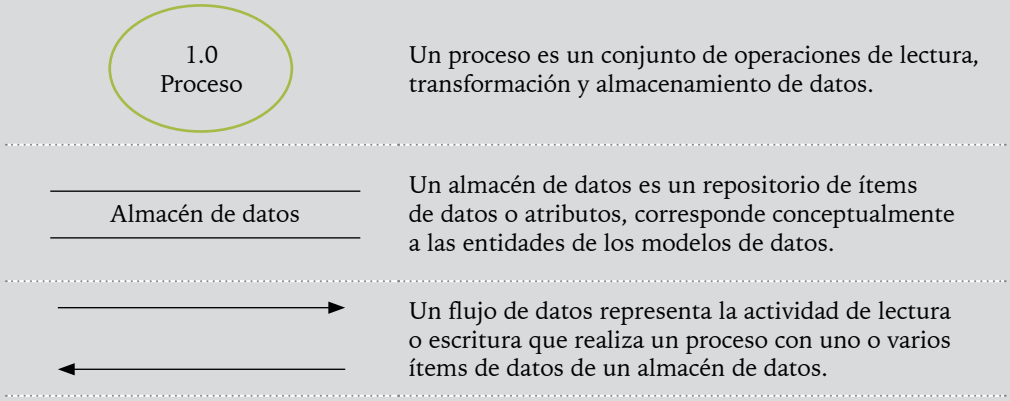

Figura 11.
Herramientas para elaborar diagramas de comportamiento.

Fuente: Pressman (2002).

Modelos de máquina de estados

Es el mismo modelo de comportamiento del sistema, en el cual este se interpreta como un grupo de estados que sirve como puente entre entradas y salidas, de manera que es la señal de entrada la que indica en cada instante un estado para la máquina y la salida está sometida al estado y a las entradas actuales (Sommerville, 2005).

Modelos de flujos de datos

Es el mismo modelo de procesos que sirve para delimitar el alcance y la serie u orden de pasos en la que se muestran las acciones que tienen lugar a partir de una entrada que se procesa, hasta la salida que establece la respuesta del sistema del sistema de información.

Los modelos de flujo de datos son una forma de suponer cómo podría ser el procesamiento de los datos a través del seguimiento y la documentación de estos por el sistema, es decir, cómo fluyen los datos a través del sistema y cómo se convierten en cada paso antes de trasladarse a la siguiente fase, representados por los diagramas de flujo de datos. Así, son estos un abrebocas para el diseño del sistema lógico y conceptual (idea lógica), y a partir del análisis esta herramienta nos sirve para representar gráficamente las actividades implicadas en un proceso al mostrar la relación secuencial entre ellas. Se recomienda para modelar la forma y las reglas en la que los datos son procesados en el sistema existente, ya que facilitan la obtención de un panorama claro del proceso y una mejor comprensión de las actividades, las relaciones y los eventos que ocurren dentro de un proceso. Asimismo, hacen fácil la identificación de los clientes y determinar sus necesidades. Por su parte, el diagrama de flujo incide para que exista una comunicación, ya que se aplica un lenguaje común que estimula la creatividad en el momento de analizar un proceso porque permite generar más ideas útiles mediante acciones orientadas a mejorar las variables de eficacia y eficiencia.

Para la construcción de los diagramas de flujo (DFD) se debe tener en cuenta las siguientes premisas:

- delimitar gráficamente el sistema;
- representar el flujo de los datos y su evolución en el sistema;
- diferir las limitaciones físicas de las lógicas.

Los elementos que intervienen en la creación de un diagrama de flujo de datos (DFD) son entidad externa, proceso, almacén de datos y flujo de datos (Sommerville, 2005).

Modelos de datos

Un modelo es una serie de instrumentos conceptuales que permite describir la estructura de los datos con referencia a las relaciones existentes entre estos, su significado y sus restricciones de consistencia. Este es uno de los elementos necesarios e importantes a la hora de emprender el desarrollo de cualquier proyecto, aunque existen varias expresiones para el modelado de datos. El modelo más utilizado es el modelo entidad relación (MER), el cual se centra en las entidades y las relaciones descritas por los datos y dota de entendimiento en el dominio de la información del problema. De esta manera, se constituye en gran ayuda para el ingeniero de *software*, ya que le sirve de base para dar inicio al diseño. Esta técnica se aplica para estructurar, organizar y documentar los datos definiendo los requerimientos de una base de datos. En el medio suele llamarse modelado de bases de datos, de manera que, básicamente, se emplean tres tipos de modelados de datos.

- *Modelo conceptual*. Es muy similar al modelo de contexto, ya que muestra de forma general y abstracta la representación global de la organización.
- *Modelo lógico*. Se encarga de mostrar la interpretación completa, lo que incluye todos los detalles de los datos.
- *Modelo físico*. Esboza la estructura que se va a poner en marcha por medio de un manejador de bases de datos (DBMS).

La arquitectura de los datos se representa mediante un modelo y utilizando un modelo entidad relación que debe contener entidades, atributos, dominios, relaciones, llaves y cardinalidad (Bentley, 2008).

Modelos de objetos

Se utilizan, básicamente, con el fin de representar tanto los datos del sistema como su procesamiento, de modo que describen objetos que son una abstracción con límites. Son de gran ayuda para mostrar cómo se clasifican las entidades en el sistema y cómo se componen de otras entidades. Este modelo nos muestra la estructura estática del sistema de la siguiente manera: objetos, relaciones entre objetos, atributos y operaciones. Así, utiliza en su sistema clases de modelos para detallarlo, tales como los que se enlistan y describen a continuación.

- *Modelo de objeto*. Representación gráfica en la que los nodos son clases de objetos que son, a su vez, relaciones entre clases.
- *Modelo dinámico*. Representación gráfica en la que los nodos son estados y los arcos son transiciones entre estados causados por sucesos.
- *Modelo funcional*. Representación gráfica en la cual los nodos son procesos y los arcos son flujos de datos.

Los programadores emplean los objetos como elementos fundamentales para dar una solución, dado que pueden construir, estructurar, agregar, cambiar, modificar o eliminar elementos y contenido con el propósito de diseñar aplicaciones y programas informáticos basados en técnicas, todas usadas con el fin de perfeccionar el modelo, ya que simplifican la transición entre DOO y la POO, en la cual se evidencia la estructura y el comportamiento de datos mediante componentes fundamentales y secundarios, tal como se enlistan a continuación.

Los componentes fundamentales son:

- *Abstracción*. Simplifica la realidad que queremos modelar para centrarnos en el comportamiento de los objetos del *software* y no en la implementación de su código.
- *Modularidad*. Propiedad que posee un sistema que ha sido subdividido en un conjunto de módulos adhesivos e inciertamente vinculados.

- *Encapsulamiento*. Proceso que restringe el acceso a los atributos y los métodos de los objetos para fijarnos en el tipo de órdenes e información que se transmite y no en su estructura y funcionamiento interno.
- *Mensajes*. Solicitudes que se hacen a los objetos para que ejecuten varias de sus rutinas.
- *Jerarquía*. Categorización o clasificación de las abstracciones.

Los componentes secundarios son:

- *Polimorfismo*. Capacidad de distintas clases para responder al mismo llamado del método, de modo que cada una lo implementa de distinta forma.
- *Persistencia*. Propiedad por la que la presencia de un objeto se propaga en su época (el autor dejará de vivir, sin embargo, el objeto seguirá vigente), o en el espacio (la ubicación del objeto se modifica, relativamente, con respecto al rumbo de creación).
- *Concurrencia*. Característica que identifica un elemento dinámico de uno no dinámico. También acepta que diferentes objetos procedan al mismo tiempo, utilizando diferentes hilos de comprobación (Sommerville, 2005, p. 165).

Diseño de sistemas

6

Un factor importante es consolidar los objetivos y la finalidad del *software*. La forma más clara de identificar sus necesidades es contar con la aprobación del documento de ERS, ya que este indica las funciones, las restricciones y las operaciones que realizará el producto. Es importante aclarar que una vez se ha obtenido la validación y la aprobación de este documento, es posible realizar cambios a los requerimientos, pero estos se deben negociar, ya que un ajuste en este sentido va a requerir que se realicen acciones de corrección a los productos que se han desarrollado y un reajuste a la programación del proyecto, así como a las diferentes actividades que realiza cada uno de los integrantes del equipo TEAM. Esto traerá como efecto la necesidad de ampliar el tiempo de entrega del producto, lo que significa que se deben asignar más recursos para cumplir con los nuevos requerimientos.

Una vez realizados estos ajustes podemos continuar con las actividades del proyecto, es decir, iniciar el proceso de diseño, ya que hemos identificado plenamente los objetivos y se ha establecido el propósito del sistema de *software*, los cuales han sido sometidos a cambios constantes al llevarlos a cabo de forma no lineal e integrarlos bajo presión de tiempo.

Las actividades del diseño deben mostrar a través de objetos las funciones y los servicios que realizará el producto, por lo cual se incluyen actores, artefactos y componentes que permiten realizar un diseño adecuado con el fin de determinar e identificar funciones que haga falta establecer y disminuir el vacío que pueda existir entre los objetos de la aplicación y la plataforma de *hardware* y *software* seleccionada.

Para elaborar los modelos de diseño y el proceso, en general, los desarrolladores tendrán en cuenta que, según la funcionalidad del *software*, deben seleccionar el lenguaje de desarrollo a utilizar. Así, por ejemplo, si es una aplicación con orientación a la web debe seleccionar entornos de desarrollo apropiados para esta en los que se pueda incorporar bibliotecas (para interfaces gráficas, estructuras de datos, etc.), así como integrar un sistema gestor de base de datos que gestione archivos en sus diferentes formatos y permita realizar acciones de administración y gestión del sistema durante el proceso de diseño de objetos. Todo esto para crear un modelo de *software* aplicando métodos más estructurados. Además, se puede aplicar el modelo de diseño de objetos, consolidando el progreso y garantizando la estabilidad del proyecto, así como aplicar acciones de forma adecuada para dar cumplimiento a los requerimientos establecidos por el cliente (Bruegge y Duttoit, 2002).

El diseño de sistemas

El diseño de sistemas es un proceso arduo y creativo que consiste en transformar el problema en una solución a través de diagramas UML, de manera que el diseñador toma como base el documento que se elaboró en la fase de análisis, es decir, el que contiene la ERS. Así, lo que hace es transformar cada uno de los requerimientos en modelos para construir la arquitectura del programa, el cual debe contener diseño de procedimientos, datos e interfaces y realizar esta actividad con base en estándares adecuados, ya que el diseño es el único camino para cumplir con exactitud los requerimientos del usuario.

En el proceso de diseño se debe realizar una serie de pasos recurrentes que el diseñador sigue en todos sus ámbitos, pues el diseño es la etapa posterior del análisis. Por esta razón él y el equipo de diseño son responsables de estimar otras configuraciones para llevar a cabo y desarrollar el sistema tal y como fue descrito en la fase de análisis, y para lo cual se utilizan dos modelos: el del diseño a través de análisis estructurado y el diseño por aplicaciones de prototipo. Según las necesidades de la organización se puede tomar cualquiera de los dos sin ningún traumatismo.

El diseño es una labor ardua y disciplinada, ya que requiere la adopción de principios fundamentales usando técnicas apropiadas y una evaluación detallada, de modo que se estructure el diseño del sistema y se tenga en cuenta el diseño de entradas, procedimientos, salidas, archivos y funcionalidades de la base de datos, así como controles, procedimientos, especificaciones y restricciones para que el avance en el desarrollo sea óptimo y logremos obtener los resultados esperados. El TEAM de diseño debe comprender que para realizar esta actividad de forma pertinente ha de guiarse por los requerimientos del sistema, planificar y generar acciones de control constante en las se verifique la calidad a través de criterios técnicos. Esto, por medio de una división del producto y a través de módulos lógicos del *software*, es decir, fraccionar en pequeñas piezas que realicen procedimientos y funciones específicas y que separen cada uno de los procedimientos de estos módulos, las cuales deben presentar una organización jerárquica que use inteligentemente el dominio entre las partes del *software* minimizando la complejidad de la conexión entre módulos y el ambiente exterior (Bruegge y Dutoit, 2002).

Actividades del diseño de sistemas

El diseño del sistema representa las funciones del *software* a partir de los requerimientos establecidos en el estándar internacional, como es UML, es decir, utilizando el lenguaje unificado para modelamiento de sistemas. Este, a través de los diagramas que contiene esta herramienta, facilita que los profesionales implicados en el desarrollo de productos de *software* logren comunicarse de forma asertiva, ya que por medio de esta herramienta se realizan diagramas tales como el modelo del negocio, casos de usos, diagramas de

iteración, modelo de clases, diagramas de actividad, diagrama de secuencia, diagrama de colaboración y diagrama de contexto, entre otros.

De acuerdo con Bruegge y Dutoit (2002, p. 192), "determinar la finalidad del diseño del sistema, es la primera acción por realizar, ya que permite registrar las funcionalidades en las que debe fundamentarse el *software*".

Varios propósitos del diseño se pueden obtener de los requerimientos no funcionales o de las restricciones de seguridad de la aplicación. Otros se podrán obtener de la información que nos brinden los interesados, sin embargo, es necesario determinarlos de forma clara y precisa para que cada una de las decisiones de diseño pueda tomarse de manera sólida siguiendo el mismo conjunto de criterios.

Los objetivos del diseño de *software* se fundamentan a partir de cinco cualidades que se enlistan y describen a continuación.

- *Confiabilidad*. Es una propiedad que debe presentar el sistema, cuyo comportamiento, en el momento de entrar en uso por parte del usuario, debe ser óptimo, es decir, proporcionar la respuesta adecuada a cada una de las peticiones que se hagan, no debe presentar fallas y debe estar disponible cuando se requiera.
- *Mantenibilidad*. Es la capacidad de adaptabilidad que debe tener el *software* en el momento que se presenten nuevos requerimientos o, en efecto cuando se requieran algunas actualizaciones. Un factor importante es que estas acciones deben realizarse a costos moderados y estar preparadas debido a la posibilidad de que se presenten algunos errores.
- *Solidez*. Es una característica que debe proporcionar el sistema en el momento de realizar procesos y procedimientos al cumplir con cada una de las funciones establecidas y presentar un comportamiento adecuado frente a las peticiones que se hagan. No debe presentar errores o fallas, por el contrario, debe presentar una estabilidad adecuada dentro de su módulo funcional.
- *Costo*. Este aspecto es determinante puesto que influye en la viabilidad económica del producto y las organizaciones realizan un análisis sobre costo/beneficio. En este se debe incluir el desarrollo del *software*, la implementación, la puesta en marcha y la administración. También permite determinar acciones procedimentales y operativas para definir el ciclo PHVA, es decir, planear, hacer, verificar y actuar en el desarrollo del sistema sin salirse del coste determinado.
- *Concepto de usuario final*. Es la valoración u opinión que tiene el usuario final frente al *software* con relación a si es amigable, si cumple con los requerimientos establecidos y si llena sus expectativas. En este aspecto se verifica la funcionalidad y la adaptabilidad entre persona/sistema operativo/maquina, de modo que el producto debe haber cumplido todas las pruebas de desarrollo.

Para iniciar el diseño de sistemas se debe haber establecido los objetivos del diseño, los cuales deben estar plasmados en el documento de requerimientos. En este deben estar los requerimientos funcionales, los no funcionales, las restricciones del producto,

los requerimientos de seguridad, de interfaz y otros determinados por el cliente. En este sentido se requiere hacer una lista de chequeo con el objeto de llevar un control y cumplir con cada una de las condiciones del *software*. Además, en este se debe tener en cuenta el rendimiento, la velocidad y el espacio en el disco duro mediante una serie de pruebas para medir la capacidad máxima de procedimientos que pueda realizar. Es importante realizar una descomposición del sistema general, pues esta acción permite reducir el nivel de complejidad del dominio y reducir así la dependencia entre las clases, el encapsulamiento de subsistemas con una interfaz unificada y sencilla.

Lo anterior permite identificar los subsistemas que se combinan. Esta actividad debe realizarse con frecuencia, ya que en ocasiones se requiere la adhesión de nuevos subsistemas, lo cual indica que son necesarias nuevas funcionalidades que implican cambios radicales en el modelo de diseño, al igual que cuando se realizan ajustes en el documento de ERS en el diseño d*el software*. También se negocian estos ajustes con miras a que el proyecto ya ha avanzado y se deben ajustar los documentos entregados como, por ejemplo, el de requerimientos funcionales, de modo que se mantengan juntos los objetos relacionados desde la visión funcional, tomando como punto de partida los casos de uso, así como estipular los subsistemas, los componentes y los demás elementos que hayan sido identificados en cada uno de ellos (Bruegge y Duttoit, 2002).

Conceptos del diseño de sistemas

Para los profesionales que intervienen en un proceso de desarrollo de *software*, es importante conocer y aplicar los conceptos del diseño, ya que estos permiten dilucidar las reglas, las normativas, las técnicas y los protocolos necesarios para entender esta área. A continuación, se enlistan y describen.

- *Abstracción*. Es una operación mental muy poderosa que permite aplicar lógica a la construcción del diseño del *software*. En esta parte se utiliza la aplicación del racionamiento y el conocimiento lógico para estructurar la relación de los elementos del diseño en el que resalta propiedades y funciones de cada una de las entidades que muestra los procedimientos que debe realizar. Esto permite establecer el nivel de relevancia de la información y que es compleja para el diseño. En general, un programa de *software* no es más que una descripción abstracta de un procedimiento o suceso que ocurre o existe en el mundo real.
- *Modularidad*. Es una acción que consiste en fraccionar o dividir un sistema de información para que queden módulos más pequeños a fin de minimizar el nivel de dificultad en el momento de dar respuesta a un problema a través de un producto de *software*. Esto con el objeto de analizar, diseñar, desarrollar y realizar pruebas de una forma sencilla, ya que para cada módulo se puede compilar y probar su propia funcionalidad aparte, sin que se afecte la funcionalidad total del sistema. Esta acción permite que el desarrollo sea menos difícil y facilita la comprensión en el momento de generar código.

En el análisis y el diseño orientado a objetos se modulariza, primero, el problema (en el análisis) y, luego, a partir de esas clases conceptuales del dominio del problema hacemos la modularizacion la solución (diseño) (Meyer, 1999).

- *Refinamiento*. Es un proceso detallado con un nivel más alto de abstracción que permite extraer detalles minuciosos para establecer el diseño inicial de la arquitectura del *software*, de manera que describe información ideal pero no provee información sobre la funcionalidad interna conforme se va desarrollando; en cada acción se analizan uno o varios procedimientos del *software* para convertirlos en instrucciones más minuciosas.
- *Arquitectura del software*. Esta representa un diseño de alto nivel en el que se conserva una estructura jerárquica por la que se relaciona cada uno de los módulos del *software*. De esta manera, se incluyen la desagregación del sistema, el flujo de control global, las políticas de manejo de errores y los protocolos de comunicación entre subsistemas (Shaw Garlan, 1996). Además, está orientado a cumplir con los requerimientos de calidad, tales como desempeño, seguridad y mantenimiento, así como sirve de guía para el desarrollo que conlleva decisiones relativas al sistema de *software*. De acuerdo con el Software Enginering Institute (SEI), la arquitectura de *software* se refiere a las "estructuras de un sistema compuestas de elementos con propiedades visibles de forma externa y las correlaciones que existen en medio de ellos" (Bass y Kazman, 2003).
- *Jerarquía de control*. Es una estructura del *software* jerárquica que presenta la relación existente en cada uno de los módulos o componentes del sistema. Esto permite realizar un análisis detallado y profundo entre estos, sin embargo, no se simbolizan los procedimientos ni se incluye en todos los modelos arquitectónicos.
- *División estructural*. La estructura del *software* se presenta de forma horizontal, se divide y muestra las actividades del *software* por escalones. En la parte superior se colocan los módulos para toma decisiones, en la parte inferior se colocan los módulos que realizan transacciones y de forma vertical se adiciona cada sección que representa una función del *software* de acuerdo con la clase, el nivel y el tipo de decisión que cada nivel jerárquico solicite.
- *Estructura de datos*. El diseño contempla la representación arquitectónica de la organización de los datos según proceso, pilas, colas, árboles, listas y bases de datos, de modo que presenta la información de manera lógica. Se tiene presente que es la estructura más utilizada y contiene datos únicos para la organización con métodos de acceso y de asociación, así como las alternativas de procesamiento. La estructura de datos es bastante importante para la representación de la arquitectura del *software*, así como la estructura del programa.
- *Procedimientos de* software. Son las acciones, los eventos y las operaciones que se realizan en cada módulo, de modo que indica la forma de procesamiento de la información y muestra detalladamente las funcionalidades de cada módulo del *software*. Se representan de forma gráfica a través de diagramas de flujo y algoritmos que proporcionan de manera lógica y específica el procesamiento de cada una de las acciones, de acuerdo con operaciones o ciclos repetitivos y la forma de organizar los datos (Bruegge y Duttoit, 2002).

El objetivo principal del diseño de sistemas de información consiste en mantener la arquitectura del *software* con el fin de optimizar la funcionalidad y la explotación de sus recursos conservando la estructura. Sus interfaces deben permitir la presentación de forma gráfica de la parte lógica del producto y que este sea intuitivo y amigable para el usuario, de manera que se tiene en cuenta la documentación del diseño de sistemas, la cual se considera una de las actividades más importantes en el diseño del *software*, ya que en esta se publican los hallazgos y las determinaciones resultantes del análisis. Consiste en registrar la información de forma clara y detallada en un documento del diseño de sistemas (SDD) y debe contemplar los aspectos que muestra la gráfica de la figura 8.

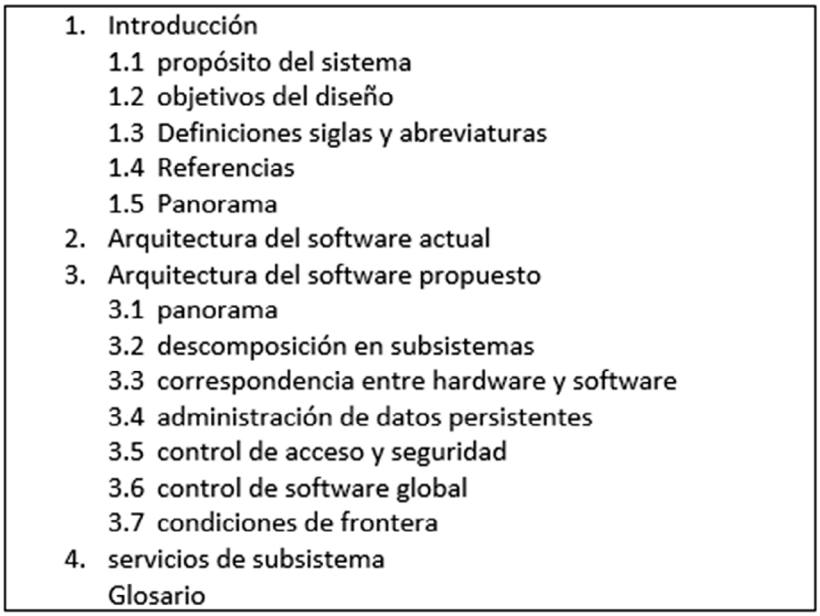

Figura 12.
Contenido del documento de diseño de *software* DSS.

Fuente: tomado de Bruegge y Duttoit (2002).

Por otra parte, la la Norma Técnica Peruana NTP-ISO/IEC 12207-2006 indica que la producción de documentos es un proceso para validar la documentación resultante en una actividad propia del ciclo de vida. El proceso contiene una serie de actividades enfocadas en la planificación, el diseño, el desarrollo, la producción, la edición, la distribución y el mantenimiento de los documentos que necesitan los involucrados, como, por ejemplo,

los gerentes, los ingenieros y los usuarios del sistema o *software*. Además, enmarca las siguientes actividades:

- implementación del *software*;
- diseño, desarrollo y producción;
- mantenimiento;
- asignación de funciones del diseño de sistemas.

Una vez terminada la fase de análisis el TEAM del proyecto debe establecer los siguientes roles:

- *Arquitecto de software*. Es el profesional encargado de liderar el proceso de diseño y establecer la arquitectura del producto, así como el responsable de la fase, ya que se encarga de la toma de decisiones, selecciona los tipos y estilos de interfaz, orienta y dirige la forma en que deben crear los diagramas al orientar la forma en cómo debe quedar la aplicación, puesto que construirá la guía para que el equipo de programadores garantice el desempeño óptimo del producto y la mantenibilidad de las aplicaciones.
- *Coordinador de arquitectura*. Se encarga de divulgar la información pertinente al equipo de diseño. En esta fase el diseño conduce la forma en la que se deben realizar los servicios que prestarán los subsistemas, mientras en la fase de implementación orienta la solidez de las APIS (interfaz de programación de aplicaciones); es el encargado de desarrollar las actividades que se describen a continuación.

Comunicación en la fase de diseño del *software*

A pesar de que las funciones del *software* están definidas y de que los profesionales cuentan con conocimiento disciplinar apropiado sobre el tema, la comunicación en el equipo de diseño tiende a tornarse difícil, a causa de temas complejos tales como el tamaño, el cambio, el nivel de abstracción, la renuencia a enfrentar problemas y a los objetivos, así como algunos discernimientos que suelen presentarse.

Administración de iteraciones del diseño

El diseño se realiza por iteraciones y modificaciones progresivas y controladas. El primer conjunto de iteraciones se desarrolla a través de lluvia de ideas, lo cual se puede realizar presencialmente o en línea. Los programadores deben darle mucha importancia a la comunicación, ya que aún no se ha definido el *software* en su totalidad. El segundo grupo de iteraciones está orientado a resolver temas complejos, de modo que permite a los desarrolladores visualizar y tratar los asuntos rápidamente con flujos de control o con modelos perpendiculares. El tercer grupo de iteraciones perfecciona anomalías del diseño que van encontrando en el transcurso del proceso (Bruegge y Duttoit, 2002).

Diseño lógico de bases de datos

Las bases de datos son un conjunto de datos lógicamente estructurados y organizados que en el momento de ser procesados se convierten en información. Estas arquitecturas deben facilitar el almacenamiento de gigantescos volúmenes de datos para, cuando sea necesario, se les pueda dar uso de forma eficiente y veraz. Por tal razón, cuando se va a crear una base de datos realizamos el diseño de un diagrama lógico y racional en el que esbozamos un esquema conceptual que permita brindar solución a un problema en una organización y debe ajustarse a una herramienta apropiada para el desarrollo de la base de datos, como son los sistemas gestores de bases de datos (SGBD). Estos son herramientas tecnológicas apropiadas para el desarrollo, la creación y la administración de los almacenes de datos, ya que permiten la implementación del sistema, pues refina y organiza la información de forma detallada cumpliendo normas estandarizadas, lo cual permite que el flujo de información se realice de forma adecuada adaptando el modelo conceptual del problema a un modelo concreto sencillo y eficaz. Esto en razón a que permite el uso de manera eficiente de los recursos para estructurar y modelar los datos adecuados en el modelo lógico que, de hecho, suele ser muy rápido si se utilizan herramientas profesionales que nos brinda la generación automática del modelo; este modelo lógico se determina como modelo entidad relación (MER).

Modelo de entidad relación

Es un diagrama que representa la estructura lógica y física de los datos con sus respectivas relaciones y muestra el nivel de cardinalidad de estas. Una vez realizado el esquema de forma física, nos valemos de un gestor de bases de datos para la elaboración del diagrama, sin embargo, a fin de entender más este tema es importante abordar los siguientes conceptos.

- *Entidades*. Son elementos de las bases de datos que cuentan con características propias en los que se hace necesario capturar y almacenar datos. Las categorías de entidades incluyen personas, sitios, objetos, eventos y conceptos que, posteriormente, se convierten en tablas.
- *Atributos*. Son la característica específica de una entidad o una fracción de información que un objeto posee o conoce de sí mismo, también llamado "variables" o "datos". Algunos atributos pueden agruparse lógicamente en súperatributos, llamados "atributos compuestos", es decir, que se componen de otros atributos.
- *Dominio*. Es la característica que posee un atributo para definir qué valores puede admitir legalmente el atributo; los valores de cada atributo se definen en tres características: tipo de dato, dominio y por omisión.

- *Relaciones*. Permiten describir la conexión lógica que puede darse entre una o más entidades, con el objetivo de comunicarse entre sí y establecer el vínculo entre estas. De esta forma, determinan el nivel de la afinidad lógica para estructurar los datos. Este componente es una pieza fundamental para el diseño del modelo conceptual porque nos facilita la documentación.
- *Llaves*. Una llave es un atributo único y sirve como identificador. Se caracteriza porque conserva un valor único para cada instancia de entidad, es decir, un campo o una unión de campos que reseña de forma original cada fila de una tabla. En una base de datos podemos encontrar varios tipos de llaves que tienen las siguientes características:
  - KP o llave primaria: es la llave principal y está conformada por uno o más atributos que permite identificar de forma única un registro o campo de una entidad. En un modelo entidad-relación, la llave primaria permite identificar las relaciones que la tabla establece con otras tablas y que van a utilizar la información de esa tabla; una llave primaria no debe permitir valores nulos.
  - Llave foránea: también conocida como llave externa, es uno o más registros de una tabla que hacen referencia al campo de la llave primaria de otra tabla. La llave foránea indica la forma cómo están relacionadas las tablas. Tanto en la llave primaria como en la llave foránea los registros deben coincidir, es decir, que tengan la misma estructura, aunque los nombres de los campos no sean los mismos.

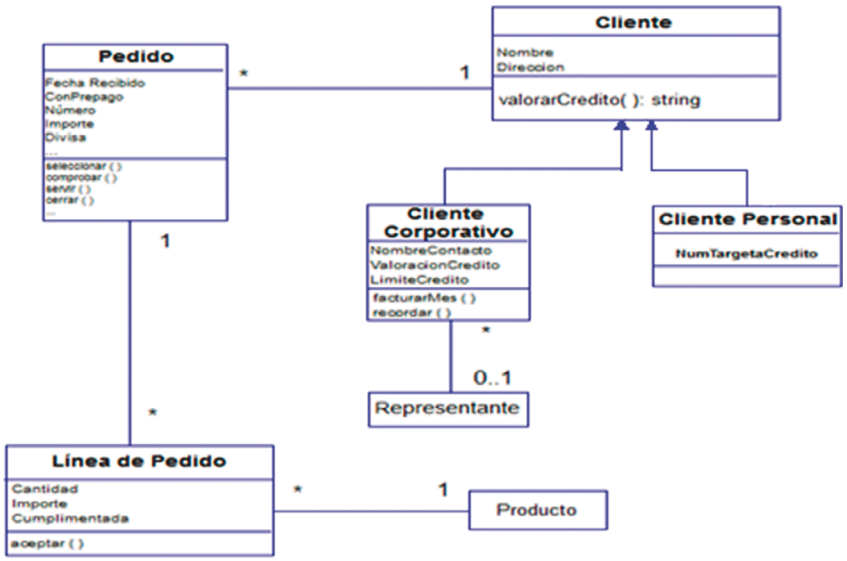

Figura 13.
Modelo entidad relación MER.

Fuente: elaboración propia en Lucidchart.

- Llave candidata: llamamos llave candidata de una relación a un atributo o conjunto de atributos que tienen la propiedad de identificar abiertamente una tupla (fila) dentro de la relación.

Las llaves son el artefacto de direccionamiento básico a nivel de tuplas en un modelo relacional, es decir, es el único modo identificado por el sistema para localizar alguna tupla específica.

- *Cardinalidad*. Representa los números de veces que una entidad puede relacionarse individualmente con otra entidad. Existen tres tipos de relaciones que pueden utilizarse entre entidades, las cuales determinan la cantidad de ocurrencias de entidad de tipo X que se pueden relacionar con una ocurrencia de entidad Y:
  - Uno a uno: una entidad se relaciona únicamente con otra y viceversa.
  - Uno a muchos o muchos a uno: determina que un registro de una entidad puede estar relacionado varias veces con otra entidad.
  - Muchos a muchos: determina que una entidad puede relacionarse con otra con ninguno o varios registros y viceversa.

Conceptos del diseño de objetos

Los avances tecnológicos han traído consigo una evolución en el diseño de *software* del diseño al pasar de modelos de flujos de datos a modelos orientados a objetos, en lo cual surge una cantidad de conceptos que es indispensable conocer, pues se convierten en fundamentos para los ingenieros que intervienen en un proyecto a fin de que puedan aplicar técnicas y métodos más apropiados y hacer del diseño una actividad más sencilla, además, porque cuentan con un conjunto de diagramas, objetos y componentes para realizar tareas. Los principales conceptos del diseño se relacionan a continuación.

- *Objetos de aplicación (de dominio) frente a objetos de solución*. Los objetos de aplicación representan conceptos del dominio que utilizan el sistema y los de solución representan componentes de apoyo que no tienen un oponente en el dominio de la aplicación.
- *Tipos, firmas y accesibilidad*. El tipo especifica la categoría de valores que puede tomar el atributo o un procedimiento; firma es el tipo de valor o parámetro de retorno de operación con el tipo de valor; la accesibilidad puede ser del atributo u operación específica si se usa o no por otras clases. Ahora, UML define tres niveles de accesibilidad: el privado, el protegido y el público. En UML a la visibilidad se le designa colocándole como prefijo el símbolo – (privado), # (protegido) y + (público) al nombre o atributo de la operación.

- *Contratos*. Se utilizan para especificar casos especiales e incluyen tres tipos de restricciones: precondiciones, especificando las restricciones que deben satisfacer el llamado antes de llamar a una operación; poscondiciones, que especifican restricciones que el objeto debe asegurar después de la operación; e invariantes que especifican restricciones de consistencia entre atributos de clase.
- *El lenguaje de restricción de objetos* UML. A los profesionales del *software* les permite crear restricciones en modelos. Estas se expresan utilizando OCL, de modo que es este un conjunto de notaciones precisas del lenguaje textual para formular restricciones que no pueden ser expresadas en la notación de diagramas estándar estableciendo una relación de dependencia (Bruegge y Duttoi, 2002).

Diseño de objetos

El diseño de objetos consiste en explicar el sistema en su comportamiento externo, como, por ejemplo, su funcionalidad, los conceptos y los requerimientos relacionando los objetos de datos y procedimientos, a fin de brindar al diseñador de *software* una pauta clara sobre la forma en la que se relacionan e interactúan los objetos y los demás componentes. Esto con miras a reducir brechas entre el problema y la máquina en las que se identifican objetos que representan ideas visibles para el usuario. Es importante dividirlos en clases para que puedan ser construidas por programadores individualmente.

El modelo de diseño de objetos se fundamenta en los siguientes puntos.

- *Abstracción*. Actividad mental que se fundamenta en lo general, de manera que omite particularidades para mejorar la apreciación y diferencia los modelos de abstracción (procedimental, de datos y de iteración).
- *Encapsulamiento de información*. Estructuración que se permite organizar datos y procedimientos de una estructura para ajustar la forma en que el objeto se pone en funcionamiento. Para esto esquiva el acceso a datos y utiliza otros medios diferentes a los definidos garantizando la integridad de estos.
- *Modularidad*. Consiste en dividir el sistema en subsistemas o módulos que contienen funciones específicas y que, al agruparlas, conforman el *software* completo (Pressman, 2007).

Actividades del diseño de objetos

Las actividades del diseño incluyen cuatro grupos de que se describen a continuación.

- *Determinación de servicios*. Consiste en localizar procedimientos faltantes, objetos necesarios para la transferencia de datos entre módulos, especificación de tipos de datos, claridad de firmas, especificación de excepciones y restricciones.
- *Clasificación de componentes*. Se buscan componentes usados y adaptados por cada módulo, se relacionan las bibliotecas de clase y los componentes extras para la organización de datos y funciones básicas, identificando y adaptando bibliotecas de clase y entornos de aplicación.
- *Reestructuración*. Incrementa la reutilización de código y cubre otros objetivos del diseño al utilizar el sistema, crear asociaciones en cada actividad de modificación y eliminar dependencias de implementación.
- *Optimización (requerimientos de desempeño del modelo del sistema)*. Contempla la modificación de algoritmos para que den respuesta a los requerimientos de velocidad o memoria, verifica rutas de acceso con el fin de activar consultas y ser eficiente, permite realizar ejecuciones, anexar atributos e iniciar la arquitectura. Por lo general, se utilizan para agrupar y mejorar el tiempo de acceso a objetos con una demora de cálculos costosos convirtiendo la respuesta en lo mejor posible (Bruegge y Duttoit, 2002).

Diseño del producto de *software*

7

Para el desarrollo de un *software* necesariamente tenemos que contar con una guía o una carta de orientación. En estos casos lo que necesitamos es un mapa o un plano arquitectónico, el cual se realiza con una herramienta CASE que nos proporciona una representación gráfica adecuada que nos orientare e indique cuáles son los componentes necesarios construir para el buen funcionamiento de nuestro producto.

Esta es una de las actividades más importantes en el desarrollo de *software*. Una vez tengamos aprobado el documento de ERS, lo que hacemos es seleccionar alguna de las herramientas CASE que nos oferta el mercado para proceder a realizar el diseño del *software* utilizando los diagramas UML que se requieran. Sin embargo, antes de identificar cuáles son los diagramas requeridos, es muy importante conocer los conceptos de diseño de sistemas, ya que a través del buen manejo de estos podremos identificar componentes y características adecuadas para la construcción del diseño. De esta manera, este se podrá realizar, así como aplicar una estructura lógica que permita concebir la estructura arquitectónica del sistema, la cual contemple desde el diagrama de contexto hasta el modelamiento de datos con miras a determinar los componentes que conforman el sistema y las actividades que desarrollará nuestro producto final. Es de gran utilidad que el TEAM equipo de proyecto dimensione la importancia de tener un buen diseño del producto e identifique plenamente cada una de las acciones y conceptos que lo componen, resaltando la importancia de esta herramienta para todas las personas involucradas en el proceso de desarrollo del proyecto (Bruegge y Duttoit, 2002).

Diagramas para el diseño de *software*

Para iniciar con la etapa de desarrollo del *software*, debe contarse con la arquitectura del producto. Esta se obtiene una vez se construya cada uno de los componentes de nuestro sistema, pero, para estar en capacidad de construirlos debemos contar con una herramienta CASE y tener pleno conocimiento de UML. A continuación, se definen los dos términos.

- UML. Es un lenguaje unificado para modelamiento de sistemas de información.
- Herramientas CASE. Es un *software* especializado para la ingeniería del *software*, es decir, a través de estas herramientas es que creamos los diferentes modelos gráficos para la interpretación de la funcionalidad del *software* facilitando la tarea de analistas, diseñadores, arquitectos e ingenieros de *software*, incluyendo a los programadores.

Algunas de las herramientas en cuestión son Power Designer, Visual Paradimg, y Rational Rose. También podemos realizar algunos diagramas en los IDE de cuarta generación como, por ejemplo, NetBeans, Visual.net, etc. Además, se encuentran herramientas *on line* para el desarrollo de estos diagramas y no tienen costo alguno, tales como Lucidchard, Genmy Model, Draw.io y Umletino, entre otras.

Cuando un ingeniero va a construir un edificio requiere una serie de planos, los cuales le sirven de guía para cumplir con los requerimientos que conlleva la construcción de la localidad y que realmente brinde servicios adecuados y no afecten el terreno ni la comunidad. Por esta razón tienen planos para cada necesidad, por ejemplo, planos para la acometida eléctrica, aguas lluvias, aguas negras, agua potable, cableado de televisión, etc.

De igual forma ocurre con los desarrolladores de *software*, quienes deben construir diferentes modelos que cuenten con una estructura lógica, lo que permita identificar cada uno de los módulos que se deben construir, identificar el papel del usuario en el sistema y la fácil interacción con este, así como reconocer cada componente, su respectiva función en el sistema y que, en efecto, cumpla con los requerimientos establecidos en la fase de análisis.

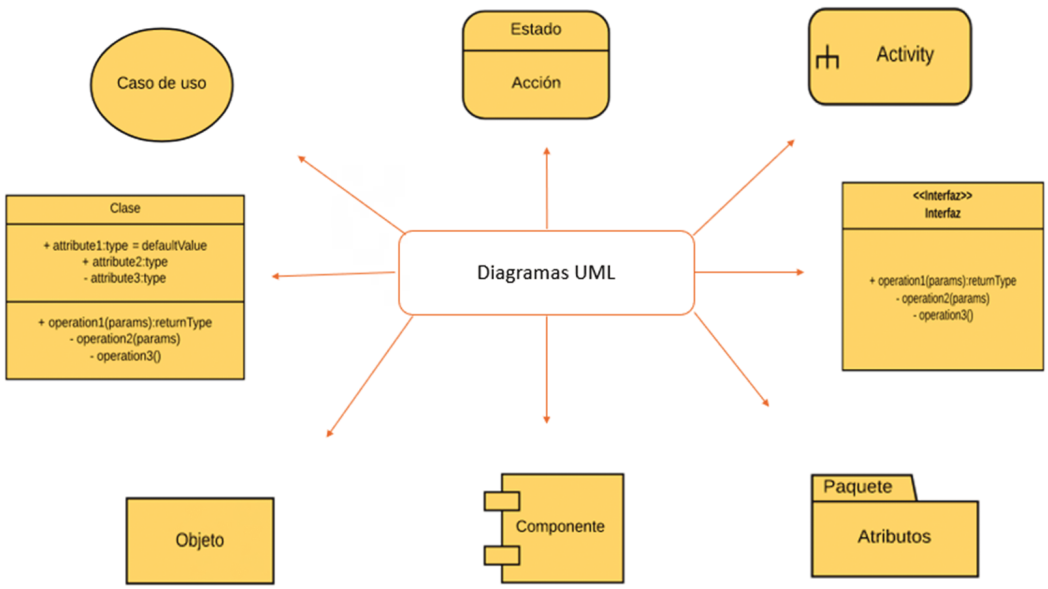

Figura 14.
Diagramas UML.

Fuente: elaboración propia en Lucidchart.

Casos de uso

Los diagramas de caso de uso representan la comunicación e interacción entre los usuarios con el *software* en el que realizan operaciones, transacciones, tareas y eventos. Permite la captura de requisitos de un usuario nuevo en el sistema, la actualización de *software* y, a su vez, sistemas participantes en el proceso, es decir, un diagrama que indica la armonía entre los actores y los casos de uso de un sistema, tal como como lo muestra la ilustración de la Figura 14.

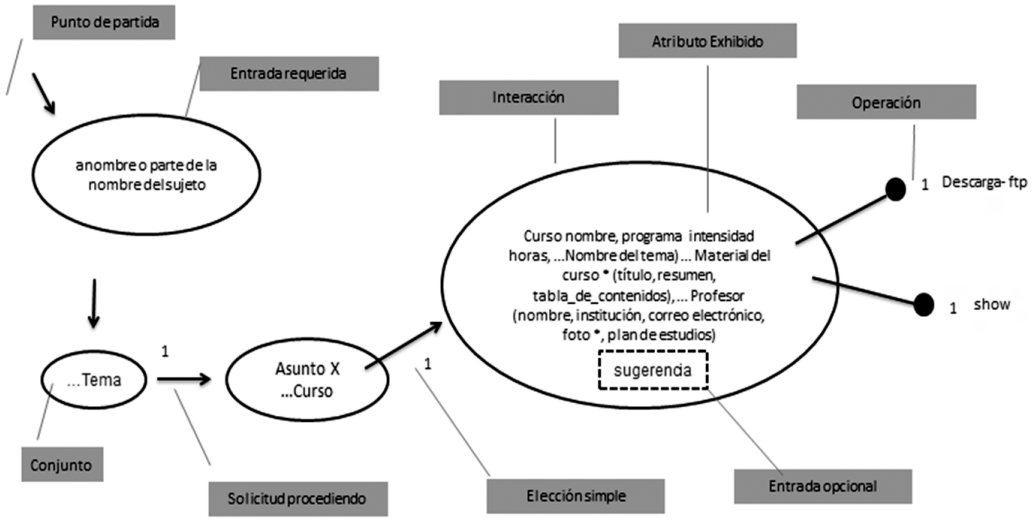

Figura 15.
Diagrama de interacción del usuario casos de uso.

Fuente:elaboración propia

Diagrama de navegabilidad

Estos diagramas representan gráficamente la forma en la que puede navegar un usuario a través de la aplicación web, ya que muestra la estructura del recorrido que se puede realizar en esta, de modo que expone claramente la interacción entre cada una de las interfaces con el objetivo de hacer la aplicación más amigable y que el usuario pueda interactuar fácilmente y así explore el contenido del sitio. A pesar de que la representación de este diagrama puede realizarse en forma de texto es más adecuado hacerlo a través de diagramas, lo que sin duda facilita la interpretación del contenido por parte del equipo de desarrollo,

de modo que domine e interprete el contenido y la forma de navegar en el hipertexto, además de que se muestre de forma adecuada según lo requiera la organización. Es importante recalcar que estos diagramas son complicados de realizar en razón a la cantidad de información que puede contener el sitio y lo difícil que es simplificar su complejidad.

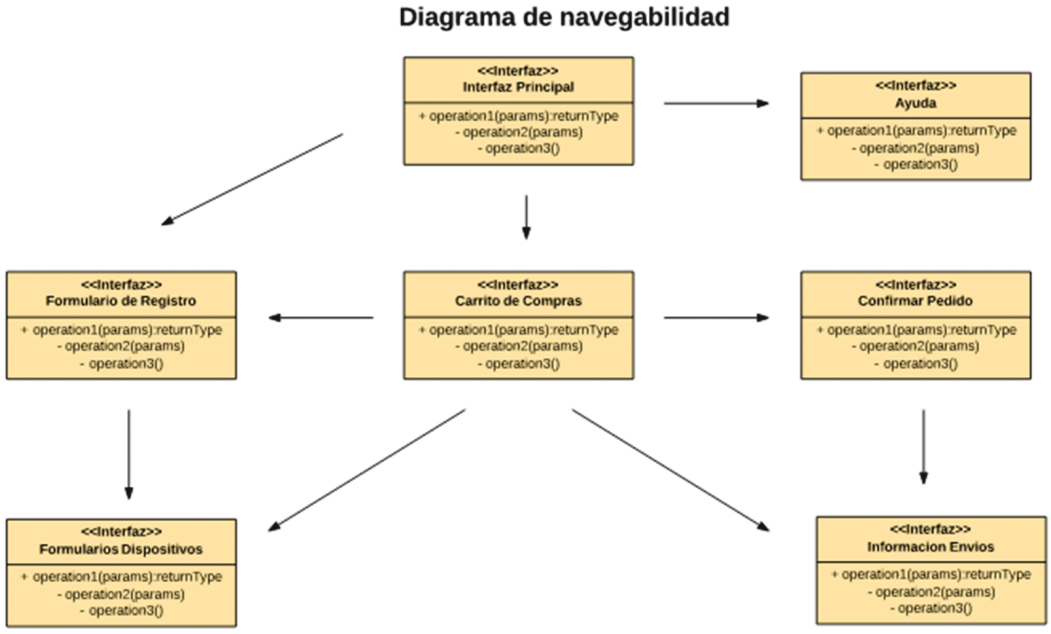

Figura 16.
Diagrama de navegabilidad.

Fuente: elaboración propia en Lucidchar.

Diagrama de clases

Estos diagramas son de estructura estática y describen clases y la forma en la que se vinculan entre sí, de manera que identifican las clases articuladas, los atributos y las subrutinas. También representan las relaciones estáticas establecidas entre ellas, identificando clases que hacen parte de otras clases, pero no deben representar los métodos que intervienen entre estas, ya que representan la forma como integran el sistema de información a nivel general.

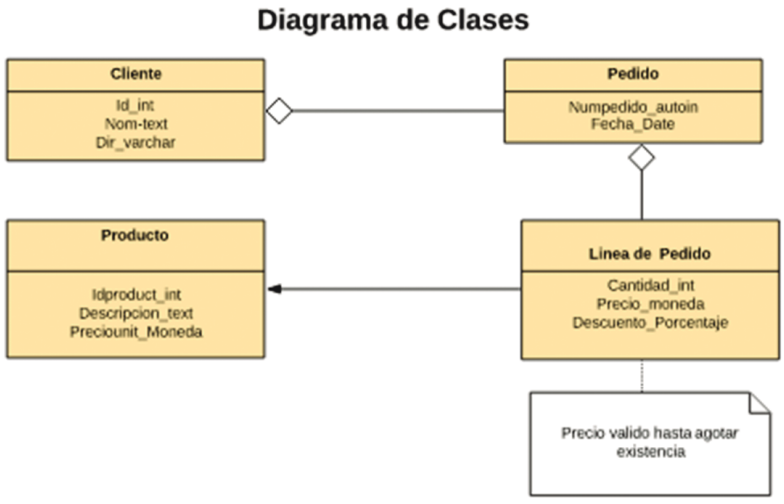

Figura 17.
Diagrama de clases.

Fuente: elaboración propia en Lucidchar.

Diagrama de secuencia

Los diagramas de secuencia representan la trazabilidad, el orden y el momento en el que se producen los mensajes que se envían a los objetos. Se interpretan por medio de líneas intermitentes verticales, con el nombre del elemento en la parte más alta. Estos diagramas nos muestran las relaciones entre objetos y el intercambio de mensajes (es decir, la forma en que se comunican) en un momento determinado.

El núcleo de tiempo se representa de forma vertical, demarcado hacia abajo, de manera que los mensajes son emitidos de un objeto a otro en forma de flechas con los nombres de los procedimientos y parámetros.

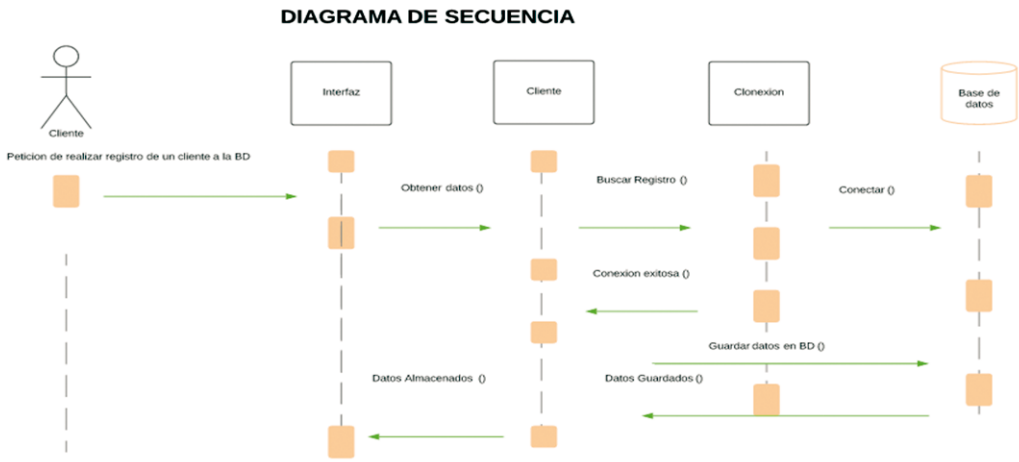

Figura 18.
Diagrama de secuencias.

Fuente: elaboración propia en Lucidchar.

Diagramas de colaboración

Los diagramas de colaboración representan las interacciones que se dan entre los objetos y sus respectivas relaciones, las cuales se encargan de actuar en una situación concreta especificando los objetos que intervienen en el intercambio de mensajes. La información que fluye es semejante a como lo presentan los diagramas de secuencia, pero reconoce la forma en la que las acciones se producen en el tiempo, mientras los diagramas de colaboración incorporan su preferencia en las relaciones entre los objetos y sus propiedades.

> En los diagramas de colaboración los mensajes emitidos de un objeto a otro se interpretan con flechas, indicando el nombre del mensaje, los límites y la secuencia del mensaje. Estos diagramas están adecuados para indicar una situación o flujo programa concreto siendo los mejores patrones para manifestar o aclarar ligeramente un procedimiento dentro de la estructura lógica del programa. (Quimbayo, 2018)

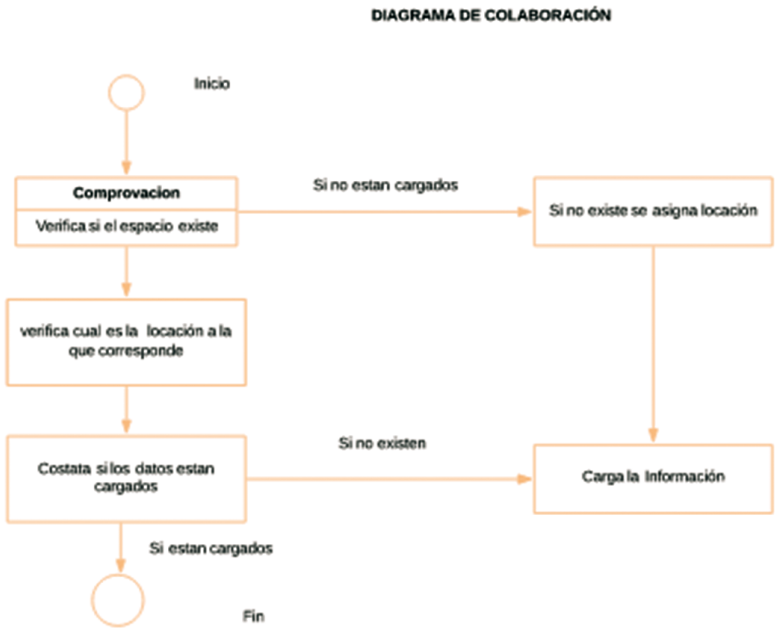

Figura 19.
Diagrama de colaboración.

Fuente: elaboración propia en Lucidchar.

Diagrama de estado

Estos diagramas representan los diferentes estados en lo que puede estar un objeto durante su ciclo. También, simbolizan los eventos y los cambios de estado en un objeto o en parte del *software* y las acciones que inciden en la variación de condiciones de un objeto.

Estos diagramas consideran los objetos máquinas de estado o robots condicionados que proporcionan una miscelánea de condiciones limitadas y que posiblemente se modifiquen según su estado a través de un estímulo pertinente en un conjunto finito. Por ejemplo, un usuario va a un banco y solicita un crédito, luego de lo cual al consultar le arrojará los siguientes estados:

- en estudio
- en revisión
- preaprobado

- no aprobado
- aprobado

Los sucesos que pueden provocar que el objeto cambie de un estado son:

- creación del objeto;
- el objeto es receptor de un mensaje;
- un usuario solicita una conexión a través de la red;
- un usuario concluye una solicitud;
- la solicitud se activa y se culmina;
- el objeto recibe un mensaje de detección, entre otros.

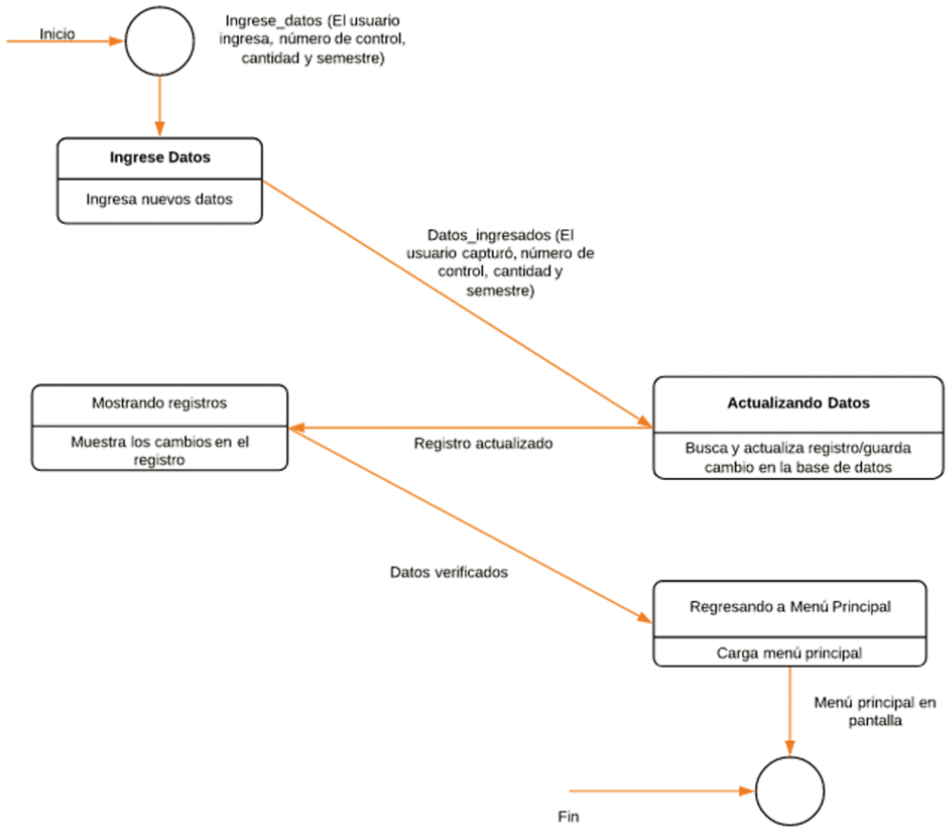

Figura 20.
Diagrama de estado.

Fuente: elaboración propia en Lucidchar.

Diagrama de actividad

El diagrama de actividad representa los eventos que ocurren en el sistema y muestra la variedad de acciones de una actividad a otra de forma detallada. Es una forma particular de los diagramas de estado y son muy similares a los diagramas de flujo procesales, con la afinidad de que todas las actividades están notoriamente ligadas a objetos y siempre están incorporadas a una clase, a una operación o a un caso de uso.

Los diagramas de actividad consideran actividades de secuencia y acciones paralelas. En el momento de realizar la acción secuencial se representa con un icono poseedor o de espera, y en el caso de las actividades paralelas no importa en que orden sean aducidas (pueden ser ejecutadas paralelamente o una después de otra).

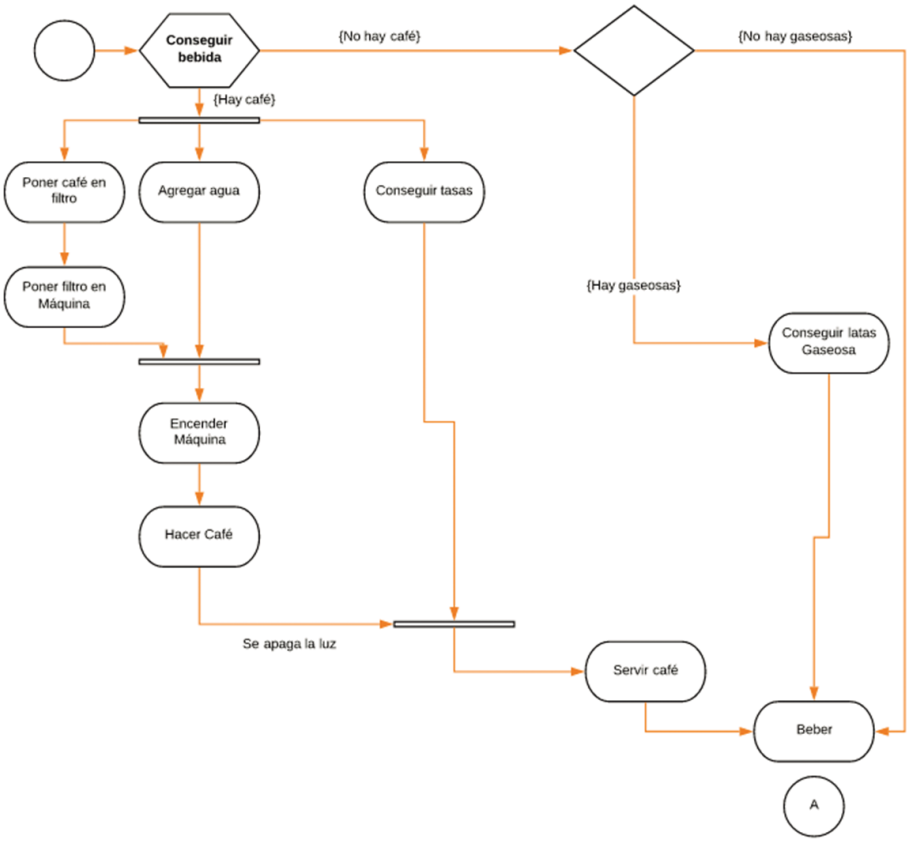

Figura 21.
Diagrama de actividad.

Fuente: elaboración propia en Lucidchar.

Estos diagramas muestran los componentes de escala superior del proceso de programación del *software*. Así, por ejemplo, en ambientes como Corba o Java NetBeans representa las tecnologías que lo integran como regiones. Estos factores pueden poseer interfaces que admiten asociaciones entre componentes tales como interfaz de aplicación, interfaz de dominio, instalaciones comunes o simplemente secciones del sistema visiblemente diferentes, además de los mecanismos de los que está compuesto como archivos de código fuente, las librerías o las tablas de una base de datos.

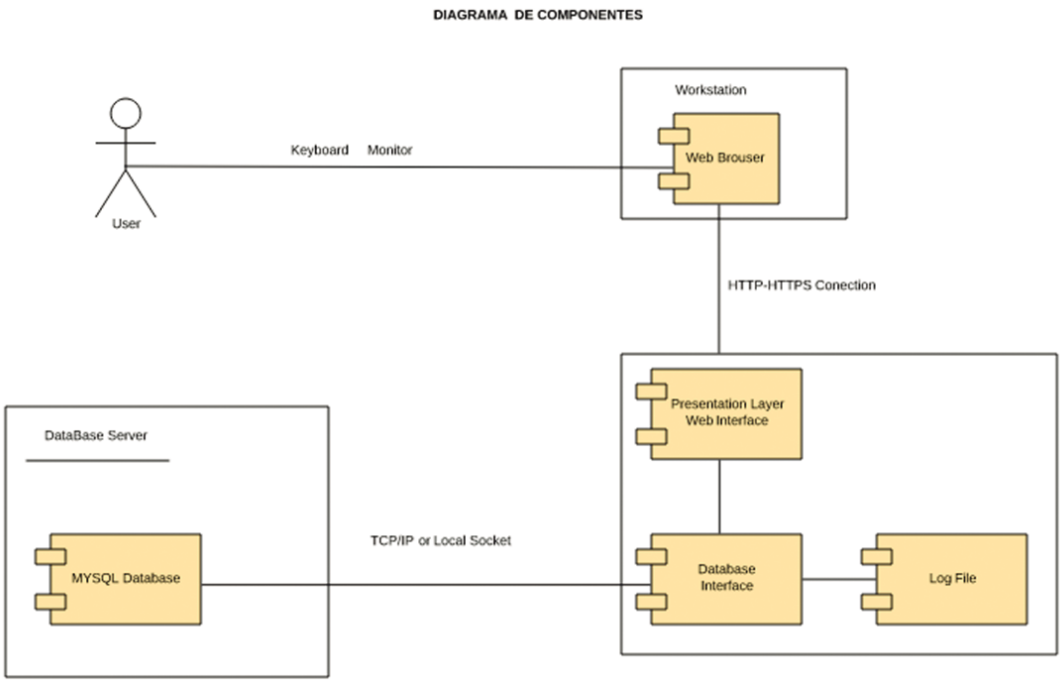

Figura 22.
Diagrama de componentes.

Fuente: elaboración propia en Lucidchar.

Estos diagramas representan en el nivel de los componentes la respectiva secuencia de relaciones, en la cual se muestra cada uno de los nodos en los que se plasman recursos físicos, tal como el ejemplo que muestra la gráfica de la Figura 25.

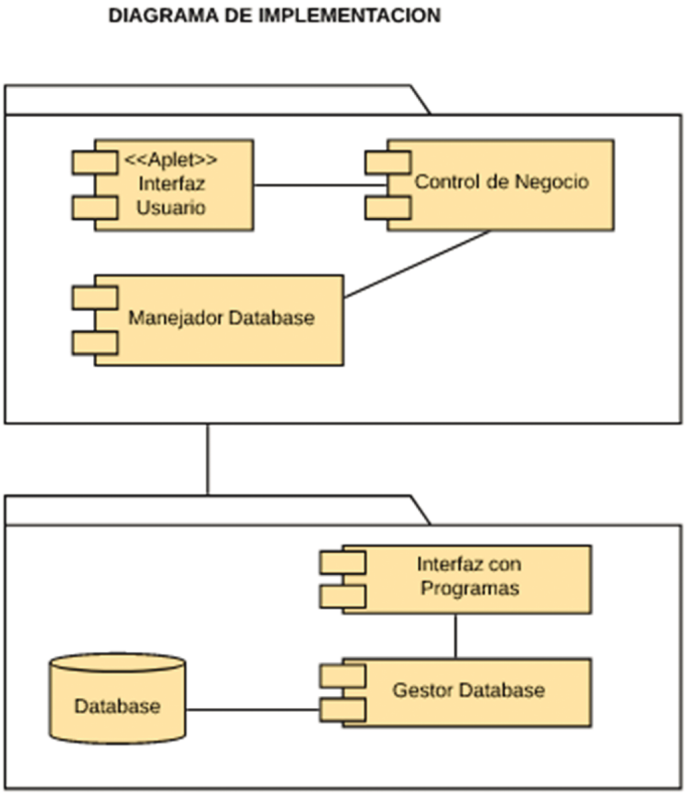

Figura 23.
Diagrama de implementación.

Fuente: elaboración propia en Lucidchar.

Diagrama de relaciones de entidad

Estos diagramas representan la estructura lógica de los atributos que puede contener la base de datos apoyados en el diseño conceptual de las de bases de datos. Toma varias nociones en el sistema de informacion y muestra las relaciones y restricciones palpables entre ellos.

Una extensión de estos diagramas es denominada “diagramas de relaciones de entidad extendida” o “mejorada” (EER) y se utiliza para asociar los procedimientos de diseño orientadas a objetos en los diagramas ER.

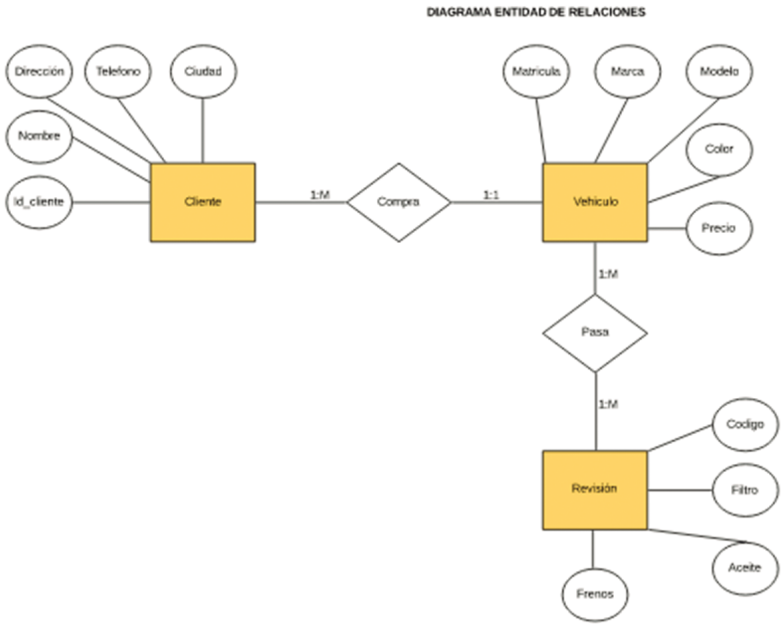

Figura 24.
Diagrama de entidad de relaciones.

Fuente: elaboración propia en Lucidchar.

Referencias

Bass, P. C. y Kazman, R. (2003). *Software architecture in practic*. Pittsburgh: Addison Wesley.

Bentley, W. (2008). *Analisis de Sistemas Diseños y Metodos* . Mexico: McGraw Hill.

Bohem, B. (2000). Software Cost Estimation with Cocomo II Publisher. Prentice Hall

Bruegge, B. y Duttoit, A. (2002). *Ingeniería del software orientada a objetos*. México: Pearson Educación.

Gonzales, H. D. (2009). *Metodología de la investigación: propuesta, anteproyecto y proyecto*. Bogotá: Ecoe Ediciones.

ISO/IEC 12207. (2008). Information Technology/Software Life Cycle Processes. edición 2 comité técnico ingeniería de software y sistemas ISO / IEC JTC 1 / SC 7 https://www.iso.org/standard

Koch, N. y Kraus, A. (2020). *The authoring process of the* UML*-based web engineering approach*. Institute of Computer Science.

Lerma Gonzales, H.D. (2001). *Metodología de la investigación propuesta, anteproyecto y proyecto*. Bogotá: ECOE Ediciones.

Hihn, J. y Habib-Agahi, H. (1991). Cost Estimation of Software Intensive Projects: A Survey of Current Practices. Conference International: Software Engineering, Proceedings,

Mendoza, M. (2004). *Propuestas metodológicas para el desarrollo de aplicaciones Web: una evaluación según la ingeniería de métodos*.

Meyer, B. (1999). *Construcción de software orientado a objetos*. Prentice Hall.

Montilva C., J. Barrios A., J. y Milagro Rivero, A. (2008). *Método de desarrollo de software para aplicaciones empresariales*. Mérida, Venezuela.

PMBOOK. (2008). *A Guide to the Project management body of knowledge*. Project Management Institute

Pressman, R. (2007). *Ingeniería del software: un enfoque práctico*. México: McGraw Hill.

Project Management Institute. (2004). *Guía de los Fundamentos de la dirección de proyectos*. Project Management Institute: Newtown Square, Campus Boulevard.

Quimbayo, A. A. (2018). *Ingeniería del software*. Bogotá: Fundación Universitaria del Área Andina.

Senn, J. (1997). *Análisis y diseño de sistemas de información*. México: McGraw- Hill.

Sommerville, I. (2005). *Ingeniería del software*. México: Prentice Hall.

Tamayo, M. T. (2002). *El proceso de la investigación científica*. México: Grupo Noriega Editores.

Yourdon, E. (1993). *Análisis estructurado moderno*. México: Prentice Hall.