Ingeniería de Software Pragmática

Guadalupe Ibargüengoitia G.
Hanna Oktaba

Agosto 2010

Facultad de Ciencias UNAM.

Contenido

Segunda parte Manuales para los roles

Apéndices

Introducción

Antecedentes

Los cursos de Ingeniería de Software, basados en los libros tradicionales (Pressman, 97), (Sommerville, 2002) o en los más recientes tales como (Braude 2003), difícilmente permiten tener tiempo para practicar los conceptos que se introducen. Los alumnos aprenden qué es un ciclo de vida y sus modelos, las fases de un proceso de desarrollo, la definición de roles y sus responsabilidades, las mejores prácticas etc. Generalmente se hacen ejercicios de cada tema y, si se decide desarrollar un sistema completo no alcanza tiempo para cubrir las fases de prueba, mantenimiento y gestión de la configuración.

Desde el 2000 hemos impartido el curso de Ingeniería de software en la carrera de Licenciado en Ciencias de la Computación de la Facultad de Ciencias, UNAM, teniendo como guía lo propuesto en TSPi (Humphrey 2000), pero hemos incorporado diversas técnicas, adaptaciones a las necesidades de estudiantes mexicanos y mejoras que son resultado de nuestras experiencias en este y otros cursos semejantes tanto de licenciatura como de maestría.

El curso de Ingeniería de software, en la carrera de Ciencias de la Computación de la Facultad de Ciencias de la UNAM, es de 7º semestre y el grupo se divide en equipos de 5 personas que se conforman por iniciativa de los alumnos. El espacio del curso, generalmente 16 o 17 semanas de 5 sesiones de una hora diaria (tres de teoría y 2 de prácticas de laboratorio), se divide en 2 ciclos de desarrollo en los cuales los equipos desarrollarán un proyecto de software. En el primer ciclo se sigue el proceso de desarrollo completo del Lanzamiento al Cierre, lo que permite enseñar y convencer a los alumnos a través de la práctica que es útil tener un proceso bien definido, entender los roles y cumplir con sus responsabilidades, así como generar los modelos, formas y otros productos documentales. En el segundo ciclo se aprecia el valor del trabajo realizado anteriormente.

Objetivo del curso

Enseñar las mejores prácticas de la Ingeniería de Software de forma pragmática, mientras se desarrolla un sistema en dos ciclos de desarrollo, trabajando en equipo con roles asignados a cada miembro del equipo.

Organización del curso

Durante el primer ciclo los alumnos aprenden por primera vez el proceso y las responsabilidades de su rol, así como técnicas para organizar el trabajo en equipo. Como apoyo, el instructor explica en horas de clase los objetivos y actividades de cada fase y rol, así como las técnicas para generar los productos. Fuera del salón de clase los alumnos practican esas técnicas para su proyecto. En reuniones del equipo se organizan para generar los productos solicitados para esa semana, llenar las formas, identificar riesgos y asignarse las tareas para la siguiente semana.

Basados en estas ideas y experiencias a lo largo de estos años, hemos generado este material que está enfocado explícitamente con esta filosofía de curso.

Estructura del material

Este libro, proporciona un proceso bien definido basado en las mejores prácticas de la tecnología orientada a objetos. Ofrece la definición de roles y sus responsabilidades. Conforma una guía práctica para aprender la Ingeniería de Software a través del desarrollo iterativo de un sistema, trabajando en equipo, pautas para la conformación de un equipo efectivo con un proceso claro y el apoyo de formas para la recolección de mediciones y el seguimiento del proyecto. Está compuesto de dos partes:

Primera parte

Capítulo 1 es una introducción a la Ingeniería de software
Capítulo 2 Desarrollo en equipo
Los siguientes capítulos del 3 al 10 corresponden a las fases del proceso de desarrollo.
Para cada fase tiene los siguientes elementos:

- Los objetivos para cada fase del proceso de desarrollo, las actividades que deben realizarse, los productos a entregar en esa fase y el rol responsable de cada producto.
- Para cada actividad a realizar se define su justificación teórica y se proporciona al menos una técnica de cómo realizarla.
- La profundización en los temas de esa fase en que se presentan los temas a los alumnos y los temas que pueden tratarse en el segundo ciclo cuando ya conocen el proceso, pero se les pueden introducir temas avanzados.
- Se incluyen preguntas que el maestro puede hacer a los alumnos en clase para saber si están entendiendo los conceptos.
- Las prácticas de laboratorio que deberán hacerse cada fase.
- Se incluye bibliografía adicional para ampliar los temas.

Segunda parte

En esta parte se encuentran manuales para los roles:

- Habilidades requeridas para ese rol.
- Responsabilidades generales de ese rol.

- Actividades que ese rol debe cumplir a lo largo de todo el proceso de desarrollo.
- Responsabilidades para cada rol en cada fase.

Apéndices

A Guión general del curso

A.1 Objetivo del curso

A.2 Calendario de las fases y actividades a realizar en cada semana del curso.

B Formas

En esta parte se incluyen 14 formas de apoyo a todo el curso.

C Orientaciones prácticas para el curso

C.1 Planteamiento del problema a resolver.

C.2 Formación de los equipos.

C.3 Calificación de los equipos.

Capítulo 1

Introducción a Ingeniería de Software

Objetivos del capítulo

- Proporcionar definiciones de Ingeniería de Software.
- Explicar qué es el software, su naturaleza y atributos.
- Enunciar los principios de Ingeniería de Software.
- Definir el Proceso de software.

1.1 Definiciones de Ingeniería de Software

Las definiciones de la Ingeniería de Software consideran diversos aspectos de lo que contempla esta disciplina:

Enfoque en el ciclo de vida de software:

- Es la aplicación sistemática, disciplinada y cuantificable del desarrollo, operación y mantenimiento de software. [IEEE Standard Glossary of Software Engineering Terminology, std610.12-1990]

Enfoque en las personas involucradas:

- Es la construcción de productos de software por grupos de personas, para que sean usados por otras. El cliente es quien solicita el desarrollo del producto y plantea el problema a resolver. El equipo de desarrollo construye y entrega el producto solicitado.

La Ingeniería de Software utiliza las teorías y características de las computadoras, objetos de estudio de las Ciencias de la Computación, para resolver problemas planteados por un cliente. (Ver figura 1). Es un campo de estudio que genera prácticas y herramientas para resolver estos problemas. Es un área que está evolucionando (se inició en 1968) para ser una disciplina completa con principios y teorías propias.

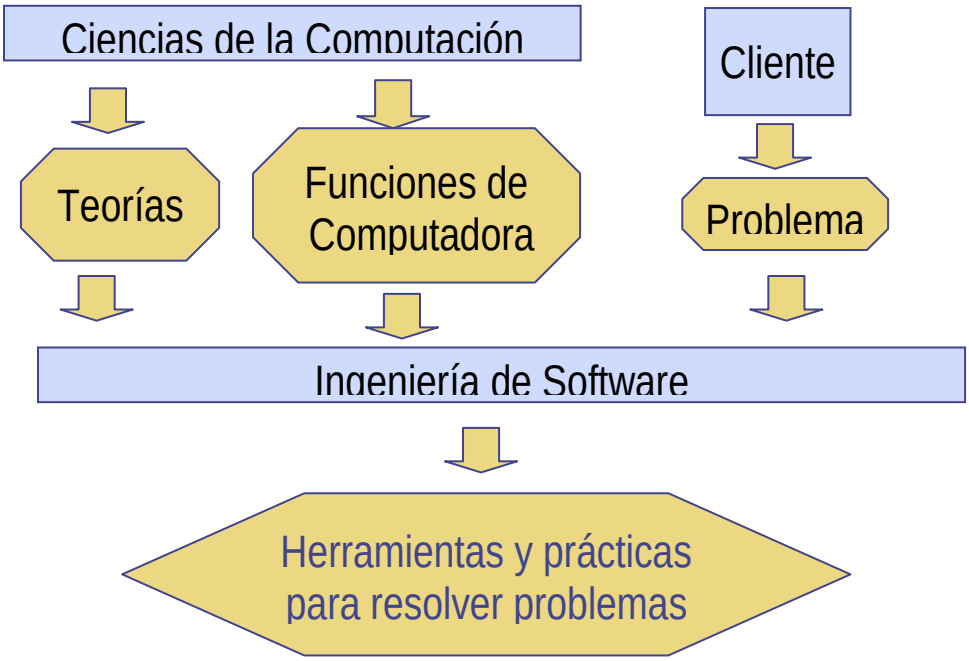


Figura 1. Relación de la Ingeniería de Software con las Ciencias de la Computación [Phleeger, 2001 p. 5]

1.2 Software, su naturaleza y características

El software es un nuevo tipo de producto que no se parece a ningún otro artefacto que sea tangible, como por ejemplo, puentes, casas, teléfonos, etc. El software tiene una naturaleza diferente, por lo que es necesario entenderla.

Se entiende por *software* al código, que resuelve un problema, generado por programas escritos en algún lenguaje de programación,. El software no es sólo el código, sino también los documentos asociados y la configuración de datos, que se requieren para que esos programas funcionen correctamente y puedan ser mantenidos [Sommerville p.5].

Un *producto de software* es la suma total de programas de computadora, procedimientos, reglas, documentación asociada y datos necesarios para la operación de un sistema computarizado [ISO/IEC 12207].

Un producto de software es un software diseñado para entregarse a un usuario. Existen varios tipos de productos de software [Sommerville p. 5]

- Software genérico (COTS Comercial Off-The-Shelf) que se venden al mercado abierto.
- Software a la medida de un cliente particular.
- Software empotrado, que forma parte de otro producto.

La naturaleza del software tiene implicaciones en su desarrollo:

- El software se desarrolla, no se fabrica en un sentido clásico [Pressman p. 11].
- El software no es tangible como los productos de otras ingenierías.
- El software es fácilmente modificable y por lo tanto corrompible.
- El software no se desgasta con el paso del tiempo, como puede pasar con el hardware, pero se puede deteriorar si al mantenerlo se le incorporan nuevos defectos al tratar de corregir anteriores. [Pressman p. 12]

Definir cuáles son las características de calidad que debe tener un producto de software no es fácil, pues depende de a quién se le pregunte: al cliente, al analista, al desarrollador, el que hará el mantenimiento, etc. Las características más reconocidos son las siguientes:

- *Funcionalidad (Functinality)*: si se comporta de acuerdo a las especificaciones de las funciones que debe ejecutar.
- *Confiabilidad (Reliability)*: si el usuario puede depender de él y si se comporta "razonablemente" aún en circunstancias no anticipadas en la especificación de requerimientos.
- *Usable (Usability)*: Si el usuario lo encuentra fácil de entender, aprender y usar.
- *Eficiente (Efficiency):* si usa los recursos (tiempo, memoria) adecuadamente y si proporciona un desempeño adecuado.
- *Mantenible (Maintainability)*: Si es fácil hacerle modificaciones tales como correcciones, mejoras o adaptaciones.
- *Portable (Portability)*: Si es posible correrlo en diversos ambientes o plataformas.
- *Reusable*: Si se pueden usar partes o todo para el desarrollo de un nuevo producto.
- *Interoperable*: Si puede coexistir y cooperar con otros sistemas.

1.3 Principios de Ingeniería de Software

Ingeniería de Software es relativamente reciente, por lo que no está tan madura como otras ramas de Ingeniería. El término fue acuñado en 1968, sin embargo, ya se han identificado una serie de principios, validados por la experiencia [Endres p.5], que deben aplicarse a través de todo el proceso de desarrollo del software. Con base a esos principios se han generado una serie de métodos y técnicas que ayudan a incorporarlos en el proceso de desarrollo de software. Estos métodos y técnicas están agrupados en metodologías. Las herramientas apoyan la aplicación de estos métodos y técnicas de forma práctica. [Fig. 2]

Figura 2. Relaciones entre principios, métodos, técnicas, metodologías y herramientas. [Ghezzi p. 44]

Los principios comprobados, según [Ghezzi p. 44], son los siguientes:

- *Generalidad*: Consiste en descubrir los aspectos más generales que existen en un problema. Este principio es fundamental cuando se trata de desarrollar herramientas y paquetes genéricos.
- *Abstracción*: Es identificar inicialmente los aspectos más importantes e ir incorporando los detalles gradualmente.
- *Modularidad*: Es dividir el problema en sub-problemas. Incluye los conceptos de cohesión y acoplamiento.
- *Incrementabilidad*: Consiste en obtener el producto de software incrementando la funcionalidad en varios ciclos (iteraciones).
- *Separación de conceptos*: Es manejar diferentes aspectos de un problema concentrándose en cada uno por separado.
- *Anticipación al cambio*: Es diseñar el software para que pueda evolucionar a nuevas versiones, que se administran de manera controlada.

El *Software Engineering Body Of Knowledge* [SWEBOK] es un esfuerzo de la comunidad de Ingeniería de Software que recoge principios, técnicas y mejores prácticas reconocidas por los académicos y profesionales del área.

1.4 Proceso de software

El Proceso de software es el conjunto de actividades que se necesitan para transformar las necesidades de un cliente en un producto de software. Permite que los desarrolladores sepan qué hacer, cuándo, cómo y quién es el responsable.

En la siguiente figura 3 se modela el Proceso de software [Oktaba] por medio de diagramas de clase de UML. Los rectángulos representan clases, la relación entre clases con un pequeño rombo significa agregación o la relación *está compuesto por*. Las líneas entre clases, significan asociaciones entre clases.

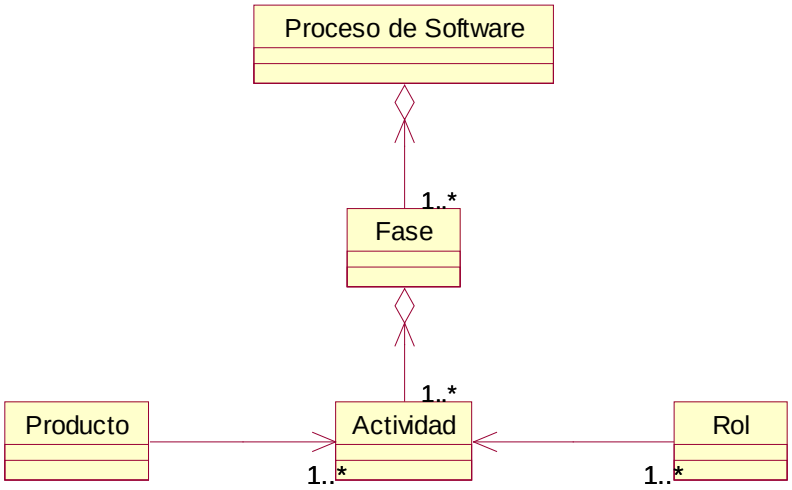


Figura 3. Modelado del Proceso de software [Oktaba]

Un **Proceso de software** lo componen una o más fases. Las fases están compuestas de al menos una actividad, que tiene asociado uno o varios productos y uno o varios roles. Un rol es responsable de al menos una actividad.

- **Fases** constituyen pasos significativos del proceso de software. Tienen un objetivo dentro del desarrollo. Para cada fase se identifican: los roles, actividades y productos que son necesarios para cumplir con el objetivo de la fase.
- **Actividades** definen las acciones que se llevan a cabo en el desarrollo del software y utilizan y/o generan los productos.
- **Productos** son las entradas y salidas de las actividades. Pueden ser documentos, diagramas de diseño, código, planes de pruebas, reportes, manuales de usuario, o conjuntos de ellos.
- **Roles** son los responsables por llevar a cabo las actividades del proceso.

En este libro el proceso de software consta de las siguientes fases, descritas en los capítulos del 3 al 10:

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo ciclo en esta fase se introducen los conceptos básicos de Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta

misma fase introduce el concepto de calidad y las *revisiones entre colegas*, como una técnica para lograr la calidad de los productos, la cual se aplicará a lo largo del desarrollo.

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado, los cuales se establecen en el Plan de pruebas del software.

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración, identificar las clases, con las que se construirá el software y describir sus relaciones a través de las vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

- **Integración y prueba del sistema** (capítulo 9). El objetivo de esta fase es la integración del software siguiendo el Plan de pruebas de integración. Se prueba que el sistema integrado cumpla con los requerimientos iniciales y se completan los manuales.

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso y del estado de la configuración.

Referencias bibliográficas

- Endres A., Rombach D. *A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories*. Pearson, Addison Wesley. 2003.
- Ghezzi C., Jazayeri M., Mandrioli D. *Fundamentals of Software Engineering*. Prentice Hall 1991.
- IEEE Standard Glossary of Software Engineering Terminology, std610.12-1990.
- Oktaba H., Ibargüengoitia G. “Software Processes Modeled with Objects: Static View”, Computación y Sistemas, Iberoamerican Journal of Computer Science, CIC-IPN, México, 1, 4 (1998), p.228-238.

- Pfleeger S.L. Software Engeneering. *Theory and Practice* 3ª edición. Prentice Hall. 2006.
- Pressman R.S. *Ingeniería del Software. Un enfoque práctico* 3ª edición. McGraw Hill 1992.
- Sommerville I. *Ingeniería de Software* 6ª edición. Addison Wesley 2002.
- SWEBOK. Guide to the Software Engineering Body of Knowledge. Versión 2004. www.swebok.org
- ISO/IEC FDIS 12207:2007 Systems and Software Engineering – Software Life Cycle Processes

Capítulo 2

Desarrollo de software en equipo

Uno de los objetivos muy importantes de este curso es aprender a trabajar en equipo de manera exitosa. Se enseñarán las técnicas para trabajar en equipo, con la finalidad de que los alumnos adquieran la experiencia que les permita incorporarse en equipos profesionales.

Objetivos del capítulo

- Formar equipos de trabajo efectivos.
- Proporcionar los elementos para llevar las reuniones semanales de los equipos.
- Definir medidas y cómo tomar las mediciones para el proyecto.
- Introducir las formas de registro de tiempo (Semana personal y del equipo).

2.1 Trabajo en equipo

El software que se necesita actualmente es cada vez más complejo. Es por esto que desarrollar software es una actividad que requiere de un equipo de personas que cooperen entre ellos.

Desarrollar software es como construir una casa. Se forma un equipo en el que participa el cliente que manda hacer la casa, en algunas ocasiones, los usuarios que la habitarán y los constructores. Los constructores tienen diversas responsabilidades o especializaciones que combinan para terminar la casa. Habrá albañiles, electricistas, plomeros y carpinteros. Todos son coordinados por el arquitecto, que define el diseño de la casa, dirige a los constructores para que hagan sus tareas en el tiempo asignado y comunica los avances al cliente.

De manera semejante, para desarrollar software se requiere de una participación organizada de varios especialistas. Tener un *equipo* no solo es reunir a un conjunto de personas para trabajar juntos, se requiere también tener un objetivo común y un *proyecto*.

Un equipo es un grupo de personas trabajando juntas de forma coordinada para alcanzar un conjunto de metas y objetivos de un proyecto.

Hay tres aspectos esenciales para el buen funcionamiento del equipo: asignación de roles con sus responsabilidades respectivas, coordinación de trabajo y comunicación. [Goldberg].

Para trabajar en equipo en el desarrollo de software se requiere:

- Dos o más personas, con responsabilidades definidas, que se comprometan a dedicar el tiempo necesario para el proyecto.

- Un objetivo común, el producto de software a desarrollar.
- Entender el producto que se desarrollará y su alcance, así como conocer al cliente y establecer comunicación con él.
- Una forma de trabajar bien definida (proceso de software) que establece qué se debe hacer, cuándo y cómo.
- Un líder que guíe al equipo y se preocupe por su buen funcionamiento.
- Una forma de comunicación dentro del equipo.
- Reuniones del equipo para establecer las tareas y darles seguimiento.

Los problemas más comunes en la conformación de los equipos son:

- Falta de identificación como equipo.
- Liderazgo poco efectivo.
- Fallas en compromisos y cooperación.
- Falta de toma de decisión.
- Falta de confianza.
- Falta de comunicación efectiva.
- Falta de un objetivo común.
- No saber qué hacer ni cuándo.

Algunas prácticas de equipos exitosos son:

- Tener un nombre, un logo, un lema y *ponerse la camiseta*.
- Comprender los objetivos del proyecto.
- Tener un proceso de trabajo común.
- Elegir al líder que coordinará al equipo.
- Planear las actividades y darles seguimiento.
- Establecer estándares para hacer los documentos.
- Comunicarse de manera libre y frecuentemente.
- Tener días fijos de reunión con agenda y duración fija.

Para que cada participante sepa qué debe hacer, se definen y asignan roles a los miembros del equipo.

Roles, responsabilidades y habilidades

Un *rol* está definido por las responsabilidades a cumplir, es el conjunto de actividades relacionadas que se asocian a un tipo de trabajo. Una persona puede ejecutar uno o varios roles y un rol puede ser cubierto por varias personas. Los roles deben estar bien definidos y ser conocidos por todos.

Para este curso se proponen 5 roles de tipo administrativo [Humphrey 2000]. Cada rol tiene una perspectiva diferente del trabajo: el equipo, el desarrollo, la planeación, la calidad y el ambiente de trabajo. El líder coordina al equipo. El administrador de desarrollo coordina la construcción del producto y asigna las tareas relacionadas con el desarrollo a otros miembros del equipo. El administrador de planeación define el plan de trabajo y le da seguimiento. El administrador de calidad es el encargado de asegurar la calidad del

producto y el administrador de apoyo proporciona el ambiente de desarrollo uniforme y controla las versiones del producto.

En la siguiente tabla, para cada rol se describen sus responsabilidades y se indica el capítulo de este libro en el que se describe el rol con detalle.

Tabla 1. Responsabilidades de los roles.

Rol	Responsabilidades	Capítulo del rol
Líder	▪ Construye y mantiene el equipo y su efectividad. ▪ Motiva a los miembros a trabajar en equipo y cumplir sus compromisos. ▪ Resuelve los problemas del equipo y de las personas que lo integran. ▪ Informa al instructor sobre el progreso del equipo. ▪ Convoca y dirige las reuniones del equipo, prepara la agenda para las reuniones.	Capítulo 11
Administrador de desarrollo	▪ Motiva y guía al equipo a seguir el proceso de desarrollo en cada una de sus fases. ▪ Es responsable por producir el producto de software con calidad. ▪ Usa y aprovecha al máximo las habilidades y los conocimientos de los miembros del equipo.	Capítulo 12
Administrador de planeación	▪ Produce un plan completo, preciso y adecuado para todo el equipo . ▪ Da seguimiento al plan cada semana. ▪ Registra riesgos que se presenten en la forma de Registro de riesgos y les da seguimiento en la forma Semana del equipo.	Capítulo 13
Administrador de calidad	▪ Coordina la definición de estándares y vigila que se cumplan. ▪ Hace un calendario para las revisiones entre colegas. ▪ Dirige las revisiones entre colegas y registra los defectos encontrados. ▪ Asegura que se corrijan los defectos.	Capítulo 14
Administrador de apoyo	▪ Proporciona al equipo las herramientas y métodos adecuados para su trabajo. ▪ Controla los cambios a los productos. ▪ Avisa a los desarrolladores cuando un cambio los afecte. ▪ Coordina las versiones del sistema.	Capítulo 15
Ingeniero de	▪ Llena la forma Semana personal.	Capítulo 16

desarrollo	▪ Asiste a la reunión semanal del equipo. ▪ Participa en las reuniones de revisión entre colegas según el Plan de calidad. ▪ Genera productos de calidad. ▪ Revisa y corrige los productos de los que es responsable. ▪ Se ajusta a los estándares del equipo.

Para que el equipo sea efectivo, cada miembro debe tener las habilidades necesarias para cumplir sus responsabilidades. La tabla 2 describe para cada rol sus habilidades requeridas y la referencia al capítulo del libro en que se detallan.

Tabla 2. Habilidades por roles.

Rol	**Habilidades**	**Capítulo del rol**
Líder	▪ Experiencia de liderazgo en grupos juveniles. ▪ Es un líder natural en los grupos. ▪ Conciliador de los intereses del grupo. ▪ Reconocimiento del grupo por su liderazgo. ▪ No conflictivo, motivador a cumplir los compromisos. ▪ Sociable y amistoso.	Capítulo 11
Administrador de desarrollo	▪ Experiencia en programación. ▪ Conocimientos de lenguajes de programación, ambientes de programación, herramientas de apoyo. ▪ Reconocimiento del equipo por sus habilidades técnicas.	Capítulo 12
Administrador de planeación	▪ Persona muy organizada, planeadora de sus actividades. ▪ Aún informalmente planea sus actividades y cree que planear es positivo. ▪ Reconocimiento del equipo por su organización. ▪ No es impositiva sino conciliadora, pero intolerante a faltas de compromiso.	Capítulo 13
Administrador de calidad	▪ Persona ordenada e interesada en la calidad del software. ▪ Reconocimiento del equipo por su interés en la calidad. ▪ Tener interés en hacer buenas revisiones a los productos. ▪ No tiene enemistades con otros miembros del equipo, lo que facilitará la organización de las reuniones de revisión.	Capítulo 14
Administrador	▪ Experto en la búsqueda de herramientas de	Capítulo 15

de apoyo	apoyo al desarrollo. ▪ Experto en cómputo e interesado por aprender y explicar al equipo nuevas herramientas. ▪ Disciplinado en el manejo de versiones de productos. ▪ Reconocimiento del equipo por su facilidad para encontrar y explicar nuevas herramientas.	
Ingeniero de desarrollo	▪ Tener conocimientos y experiencia en programación, estructuras de datos y bases de datos.	Capítulo 16

Comunicación en el equipo

La comunicación externa e interna de un equipo es esencial para su buen funcionamiento.

La comunicación externa del equipo con el instructor se lleva a cabo por medio del líder. El líder informa al instructor lo que ha hecho el equipo, los problemas que surgen y los riesgos. El instructor da seguimiento al estado del equipo. En proyectos reales, el líder del proyecto también establece la comunicación con el cliente para ir refinando los requerimientos del producto e informarle sobre los avances.

La comunicación interna del equipo sirve para que todos conozcan el estado de trabajo de los demás. Para esto se establecen reuniones semanales del equipo. En estas reuniones se revisan los avances del trabajo y se toman acuerdos.

Coordinación del equipo

La estructura del equipo está influida por el proyecto, la coordinación del grupo y las necesidades de comunicación. *La estructura es la forma en que se organizan las personas, se comunican, se asignan responsabilidades e informan los avances.* [Goldberg]. Existe todo un espectro de tipos de estructuras para organizar los equipos: desde las muy jerárquicas a las muy democráticas.

Los equipos jerárquicos se organizan como un árbol, donde en la raíz está la autoridad y en niveles inferiores los trabajadores técnicos. Esta estructura es común en organizaciones grandes con niveles de autoridad bien definidos. La idea central es que las autoridades toman las decisiones y delegan el trabajo a los niveles inferiores.

En equipos democráticos todos trabajan al mismo nivel y son iguales. Los equipos de estudiantes son ejemplos claros de esta estructura democrática. En procesos de desarrollo de software, llamados ágiles, también se sigue una estructura democrática.

2.2 Reuniones semanales del equipo

Para establecer una comunicación efectiva existen métodos *informales,* en los que se establece la comunicación instantánea cuando se necesita, y *formales,* a través de reuniones semanales del equipo. Se propone seguir ambas formas de comunicación.

Al definir el equipo de trabajo se establece el día de la semana y la hora en que se tendrán las reuniones de todo el equipo.

Los objetivos de estas reuniones semanales son los siguientes:
- Reportar avances.
- Acordar compromisos.
- Identificar problemas y buscar soluciones.

Para que estas reuniones sean efectivas se debe procurar:
- Establecer el día, hora y duración máxima de cada reunión.
- El líder propone la agenda de cada reunión.
- La coordinación de la reunión la lleva a cabo el líder del equipo.
- Se debe cumplir la agenda en todos sus puntos.
- Establecer el compromiso de asistencia de todos los miembros.
- Dejar por escrito, los compromisos acordados en la minuta que se envía a todos los miembros o se deja en un lugar que puedan consultar todos (correo electrónico, un pizarrón, una carpeta del proyecto, etc.).

A continuación se desglosa en un procedimiento cómo preparar y llevar a cabo una reunión semanal del equipo y el esquema de la agenda típica.

Procedimiento típico para realizar las reuniones:
1. El líder convoca a la reunión y manda la agenda.
2. Al iniciar la reunión, el líder lee la minuta de la reunión anterior y la agenda actual, permitiendo proponer modificaciones.
3. Se pasa lista de asistencia y se anota la hora de inicio real.
4. El líder dirige la reunión siguiendo la agenda y tratando de no desviarse de ella. Todos los miembros entregan su forma *Semana personal* (Ver 2.4) de la semana anterior al Administrador de planeación.
5. Cada rol informa de sus avances y problemas.
6. Se informa del avance de las tareas de desarrollo.
7. Se acuerdan nuevos compromisos.
8. La reunión termina cuando se agota la agenda y se han reportado todos los avances, problemas y se han acordado compromisos para la siguiente semana. Los problemas se registran en la forma *Registro de Riesgos* (Ver 2.4). Los compromisos de cada quien para la siguiente semana se anotan en su forma *Semana personal* (Ver 2.4) de la siguiente semana.
9. Se anota la hora de terminación de la reunión.

10. La agenda y la minuta de la reunión se incorporan al depósito del proyecto y se entrega o envía a todos los miembros del equipo.

Esquema de la agenda típica para las reuniones de los equipos:

1. Revisar la agenda propuesta.
2. Revisar el cumplimiento de los acuerdos.
3. Informar los avances y problemas de cada rol:
    a. Administrador de desarrollo
    b. Administrador de planeación
    c. Administrador de calidad
    d. Administrador de apoyo
    e. Líder del equipo
4. Informar las tareas como Ingenieros de desarrollo.
5. Todos los miembros entregan su forma *Semana personal* de la semana pasada al Administrador de planeación para que resuma los datos en la forma *Semana del equipo*.
6. Asignación de tareas para la siguiente semana en la forma *Semana personal* de la siguiente semana.
7. Registro de problemas y propuestas de soluciones en la forma de *Registro de Riesgos*.

Esquema de la minuta típica de las reuniones del equipo

1. Nombre del equipo, fecha, hora de inicio, hora de terminación.
2. Lista de asistentes.
3. Lista de productos terminados de esta semana.
4. Asignación de tareas y responsables para la siguiente semana.
5. Identificación de problemas y propuestas de soluciones.

Cada semana se envía al instructor la minuta de la reunión del equipo.

2.3 Medidas y su papel en la Ingeniería de Software

Las medidas se necesitan para conocer, entender, comparar y mejorar el desarrollo de software con bases cuantitativas. La Ingeniería de Software requiere de medidas tanto para los procesos como para los productos. Para entender las capacidades de cada miembro del equipo, cada ingeniero debe tomar datos de su desempeño en cada fase del desarrollo.

- Una *medida* (measure) proporciona un indicador cuantitativo sobre la cantidad o tamaño de un atributo de un producto o proceso.
- Una *medición* (measurement) es la toma de una medida.
- Una *métrica* es el conjunto de mediciones que sirven al equipo para entender su desempeño, si se va mejorando o empeorando con respecto a otras mediciones.

El proceso de medición permite al equipo evaluar la información recaudada del estado actual y compararlo con las metas planteadas, ayuda a tomar decisiones para reducir las diferencias entre lo planeado y lo obtenido en un proceso de mejora continua. [Ebert]

Según Personal Software Process [Humphrey 97] se deben tomar medidas individuales del tiempo dedicado a las actividades de desarrollo para ayudar a los ingenieros de software a controlar, manejar y mejorar su trabajo. Se realizan la toma de medidas personales de tiempo, al efectuar cada una de las tareas de desarrollo, y del tamaño de los productos y estas se registran en una forma de Registro de tiempos.

Team Software Process propone que cada miembro del equipo resuma su trabajo de la semana en la forma *Semana personal*. A partir de los datos en estas formas de todos los miembros, el administrador de planeación llena la forma *Semana del equipo* que resume las tareas efectuadas esa semana por el equipo, los tiempos en realizarlas, las tareas completas y los riesgos detectados.

Estas mediciones se convierten en datos históricos para futuros ciclos y proyectos. El uso de los datos históricos permite mejorar la planeación de nuevos proyectos al hacer estimaciones de tiempos, tamaños y calidad basados en lo sucedido en proyectos anteriores. Estos datos históricos permiten también prevenir los riesgos y hacer propuestas de mejoras.

2.4 Formas Semana personal y del equipo.

Como parte de apoyo al aprendizaje en este curso teórico-práctico se han tomado como base algunas formas propuestas en el libro de TSPi adaptándolas y adecuándolas a las necesidades del curso.

Forma Semana personal

El resumen de los productos trabajados cada semana por cada miembro del equipo, se registra en la forma *Semana personal*. En el encabezado de esta forma se registra el nombre de la persona, el de su equipo, la semana que se reporta y el ciclo de desarrollo en que se está trabajando.

Su objetivo es registrar los productos en que se trabaja cada semana indicando el tiempo que se les dedicó (aproximadamente), el tamaño de ese producto y el estado de avance del producto en esta semana. Estos datos se registran en una tabla que tiene los siguientes campos:

Productos de desarrollo generados. Se lista los productos en que se trabajó esta semana.
Tiempo dedicado. Es el tiempo dedicado en esta semana a cada producto en todas las sesiones en que se trabajó en él. Cada vez que se trabaja en un producto se puede anotar en esa columna el tiempo invertido y sumar esos tiempos al final de la semana y marcar el total.

Tamaño del producto. Se indica el tamaño del producto en unidades que corresponden al tipo de producto, pueden ser páginas, líneas de código, diagramas, etc.

Productos de desarrollo generados	**Tiempo dedicado**	**Tamaño del producto**

Forma Semana del equipo

En la reunión semanal del equipo, cada miembro informa de sus avances entregando su forma Semana personal. El administrador de planeación recoge todas estas formas y hace un resumen del avance del equipo en la forma *Semana del equipo* que entregará al instructor cada semana.

El encabezado de esta forma indica el nombre del equipo, la semana y ciclo que se reporta.

En ella se registran el estado actual del avance del proyecto haciendo un resumen del avance de todos los miembros. Consiste en 3 tablas en las que se registra:

Productos de desarrollo generados esta semana
En esta tabla se resume los productos en que trabajó el equipo en esta semana, indicando para cada producto la suma de los tiempos empleados por todos los miembros que trabajaron en él, la suma de los tamaños generados por todos los miembros del equipo para el producto es esta semana.

Estos datos se toman de las formas personales.

Productos de desarrollo generados esta semana	**Tiempo dedicado**	**Tamaño del producto**

Datos globales del proyecto en esa semana:

En esta tabla se registran los tiempos dedicados por los miembros del equipo a las actividades de desarrollo.

Tiempo que se ha dedicado al proyecto en esta semana. Se obtiene de sumar todos los tiempos reportados por los miembros para todos los productos en esta semana.
Tiempo del proyecto en este ciclo hasta esta semana. Se suman el tiempo dedicado en esta semana al total de la semana anterior.
Productos terminados de esta fase en esta semana. Se indican cuáles productos se terminaron esta semana.

Esta forma incluye una tabla para dar seguimiento a los riesgos identificados en esta semana y su estado. En el capítulo 3 se revisará que son los riesgos y cómo manejarlos.

Seguimiento de riesgos	**Estado**

Forma Semana del equipo del 2º Ciclo

Esta forma cambia en el segundo ciclo puesto que se aprovechan los datos recolectados en el primer ciclo y se pueden hacer estimaciones de tiempo y tamaño para los productos. Esta tabla es como sigue.

Productos de desarrollo generados	**Tiempo estimado**	**Tiempo actual**	**Tamaño estimado**	**Tamaño actual**

Además se llena esta tabla en la que se indica, según lo estimado cuánto tiempo menos (o más y se pone negativo) se empleó esta semana y cuánto se ha ganado en el ciclo sumando lo ganado hasta la semana anterior (forma semana del equipo de la semana pasada), mas lo ganado de esta semana (renglón de arriba).

Tiempo ganado en esta semana	
Tiempo ganado en este ciclo a la fecha	

Referencias bibliográficas

- Ebert Ch., Dunke R., Busndschuh M., Schmietendorf A. *Best practices in Software Measurement. How to use metrics to improve project and process performance*. Springer 2005.
- Goldberg A., Rubin K. *Succeeding with Objects: decision framework for project management*. Addison Wesley 1995. Cap. 12.

- Humphrey Watts S., *Introduction to Personal Software Process*. SEI Series in Software Engineering Addison Wesley 1997.
- Humphrey Watts S., *Introduction to Team Software Process*, SEI Series in Software Engineering, Addison Wesley, 2000.
- Pfleeger S.L. *Software Engeneering. Theory and practice* 3ª edición. Prentice Hall. 2006.
- Tomayko J. E., Hazzan O. *Human Aspects of Software Engineering*. Charles River Media, Computer Engineering Series. 2004.

Capítulo 3

Fase de Lanzamiento

Objetivos del capítulo:

Explicar los objetivos y las actividades de la fase de Lanzamiento.

- Qué son los objetivos del equipo, del producto, personales y por rol.
- La necesidad de definir estándares para los productos.
- El concepto de riesgos y su manejo.

3.1 Fase de Lanzamiento: objetivos, actividades y productos.

Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Objetivo:

O1. Organizar el equipo y asignar roles.

Antes de iniciar las actividades técnicas de un proyecto de desarrollo de software hay que realizar muchas actividades previas. Por ejemplo, definir los objetivos del equipo y del proyecto e identificar los objetivos personales en el proyecto.

Actividades:

A1.1 Discutir los objetivos del equipo, del producto de software.
Para uniformizar los intereses del equipo se plantean cuáles serán sus objetivos: como equipo y del producto.

A1.2. Entender los roles, definir responsabilidades y objetivos de cada rol.
Se revisan las responsabilidades de cada rol y cada quien especifica cuáles serán sus objetivos en ese rol y sus objetivos como persona en el proyecto.

Productos:

Documento de objetivos del equipo, producto, personales y de rol.
Minuta de la reunión.

Objetivo:

O2. Tener un estándar para los documentos generados por equipo**.**

A fin de que todos los productos tengan un mismo formato independientemente de quien los realiza y para mejorar la comunicación en el equipo, se establece el estándar para los documentos que todos respetarán.

Actividad:

A2.1 Establecer el estándar para la documentación.
Se acuerda entre todos el estándar que deberán seguir todos los documentos.

Producto:

Estándar de documentación.

Objetivo:

O3. Manejar los riesgos del proyecto.

Para prevenir asuntos no deseados y disminuir su posible impacto sobre el desarrollo del proyecto, se identifican y manejan los riesgos desde el inicio.

Actividad:

A3.1 Identificar los riesgos del proyecto.
Para manejar los riesgos se identifican las situaciones no deseadas que podrían suceder en el transcurso del proyecto y la forma en que podrían eliminarse o disminuir su impacto.

Producto:

Forma de Registro de riesgos

Formas a entregar en esta fase:

- Semana personal
- Semana del equipo
- Minuta de la reunión

Resumen de productos a entregar:

Producto	Responsable
Documento de objetivos	Líder del equipo
Estándar de documentación	Administrador de calidad
Registro de riesgos	Administrador de planeación

3.2 Definición de objetivos

Para constituir un equipo exitoso, se requiere de un objetivo común. Una vez establecidos el nombre del equipo, logo y lema, lo siguiente es definir el objetivo del equipo.

Ejemplos de objetivos del equipo:

- Aprender buenas prácticas de la ingeniería de software para desarrollar productos de software de calidad, etc.
- Constituir un equipo bien organizado e integrado.
- Generar un producto de calidad.
- Generar todos los productos del proceso de desarrollo.
- Tener una administración buena y efectiva.
- Concluir a tiempo el trabajo.

También se definen los objetivos que se propone el equipo para el producto que se generará. Los objetivos para el producto pueden ser:

- Que el producto cumpla con lo solicitado por el instructor.
- Que sea eficiente.
- Que sea útil para el usuario, etc.

Estos objetivos se establecen por consenso.

Cada miembro del equipo propone sus objetivos personales que podrían ser:

- Acreditar la materia de Ingeniería de Software
- Aprender a trabajar en equipo.
- Colaborar en el trabajo.
- Ser disciplinado.
- Cumplir con lo que se le asigna.
- Entregar a tiempo los productos de los que es responsable.
- Comprometerse a trabajar el tiempo necesario, etc.

A partir de lo descrito en el capítulo de cada rol (capítulos del 11 al 15) se identifica lo que se espera de él y lo plantea como los objetivos de su rol.

3.3 Estándar de documentación

Al trabajar en equipo es indispensable que se establezca un acuerdo en la forma en que se entregarán los documentos que se generan durante un proyecto. La finalidad es tener una imagen uniforme y facilidad de su identificación. Para esto se define un estándar al que se apegará el equipo para generar sus productos. El estándar incluye: el nombre del editor de

texto, tipo y tamaño de las letras a usar en los títulos, subtítulos y texto, definición del encabezado de los documentos, que debe tener: título del documento, nombre del equipo, responsable, fecha, versión, número de página.

Los estándares deben ser claros y simples y que todos se comprometan a respetarlos. Se pueden seguir estándares publicados por asociaciones internacionales como IEEE, ISO, etc.

3.4 Manejo de riesgos

Un riesgo es un evento no deseado que puede tener consecuencias negativas en el proyecto. Es importante identificar los riesgos desde el inicio para tomar decisiones de cómo manejarlos. En la reunión del equipo, todos se ponen a imaginar los riesgos que podrían suceder durante el ciclo de desarrollo y éstos se anotan en la forma de Registro de riesgos.

Para manejar un riesgo identificado se debe analizar la probabilidad de que suceda, evaluar su impacto, proponer alguna estrategia para manejarlo y en su caso efectuarla.

El manejo de riesgos implica darles seguimiento a lo largo de todo el proyecto, llevando a cabo acciones para minimizarlos, evitar o contenerlos.

Hay varios tipos de riesgos:

Indirectos: no se tiene control sobre ellos, por ejemplo: falta de dinero del cliente, enfermedad de un miembro, ocurrencia de una catástrofe natural etc.

Directos: se tiene cierto grado de control sobre ellos, por ejemplo: falta de planeación adecuada, requerimientos no claros, falta de conocimientos del lenguaje de programación a usar, etc.

Para identificar los riesgos hay que detectar qué los puede producir. Hay varias fuentes de riesgos en los proyectos de software, como por ejemplo:

- **En la administración del proyecto**: retraso en tiempo, aumento del costo, cambio o disponibilidad del personal, cambio de requerimientos, aumento del alcance, equipo no integrado, etc.
- **Técnicos**: requerimientos no claros, problemas de diseño, desconocimiento de la tecnología de implementación, herramientas insuficientes o inexistentes en un momento dado, ambiente de desarrollo inestable, etc.
- **Negocio**: cambios estratégicos de la compañía, que quita recursos y apoyo al proyecto, etc.
- **Otros:** que se identifican de antemano al iniciar el proyecto.

Los riesgos tienen 2 atributos que se deben identificar:

- **Probabilidad del riesgo**: es el nivel de certeza de que el riesgo ocurra. Los indicadores de probabilidad pueden ser: alta, media o baja.

- **Impacto del riesgo**: es la consecuencia asociada a que ocurra el riesgo. Los indicadores del impacto pueden ser: alto, medio, bajo.

Para manejar el riesgo existen varias estrategias como son:
- **Aceptarlo**: vivir con él y tenerlo presente en el proyecto.
- **Evitarlo**: llevar a cabo acciones para eliminarlo.
- **Mitigarlo**: tratar de reducir su impacto o probabilidad.
- **Contenerlo**: hacer un plan alternativo para minimizar su impacto.
- **Prevenirlo**: tratar de que no suceda.
- **Transferirlo**: identificar quien se pueda encargar de manejar el riesgo.

Formas de Registro de riesgos.

Para registrar los riesgos se propone la forma de Registro de riesgos. El responsable de llenar esta forma es el administrador de planeación. Esta forma se llena en equipo desde el inicio del proyecto. Cualquier miembro del equipo puede detectar un riesgo y lo comunica al administrador de planeación para que lo registre.

La forma tiene un encabezado semejante a la Semana del equipo.

Forma de Registro de riesgos

Equipo	**Semana**	**Ciclo**

Riesgo	**Probabilidad de ocurrencia**	**Impacto**	**Estrategia para manejar el riesgo**

Consiste en una tabla en que se llenan las siguientes columnas:
- **Riesgo** que se detecta**.**
- **Probabilidad de ocurrencia.** Puede ser alta, media o baja.
- **Impacto.** Puede ser alto, medio o bajo.
- **Estrategia para manejar el riesgo**. Puede ser: controlarlo, contenerlo, inexistente, turnado a, etc.

Además de identificar los riesgos al inicio del proyecto, hay que darles seguimiento. La forma Semana del equipo tiene una tabla en la que semanalmente se les da seguimiento. Se revisan los riesgos identificados para ver que tanto afectaron el trabajo de la semana y se registra su estado al finalizar la semana. Pueden identificarse nuevos riesgos e indicar su estado.

Seguimiento de riesgos	**Estado**

3.5 Profundización de los temas para el segundo ciclo

Objetivos y su medición.

Una razón de establecer objetivos es comprobar si se cumplen o no. Para esto es necesario tener una forma de cuantificar si se cumplieron a cabalidad.

En el segundo ciclo, se revisa si subjetivamente se cumplieron los objetivos enunciados en el primer ciclo. En el segundo ciclo se pueden proponer los mismos objetivos pero con algunas formas de medir su cumplimiento. Si los objetivos anteriores no eran muy realistas o fueron demasiado generales, se proponen nuevos, que pueden ser medidos según la experiencia acumulada y los datos recopilados en las formas que se llenaron en el primer ciclo.

Por ejemplo:

Objetivos del equipo:

- *Concluir el trabajo a tiempo*

Medición posible: Haber entregado el software en la semana asignada por el instructor.

- *Producir un producto de calidad*

Medición posible: En la forma de **Informe de mediciones y sugerencias de mejora** (Ver Formas) obtener menos de 5 defectos por KLOC en 90% de los productos.
Otra medición posible: Obtener más de 3 en la forma de **Evaluación del equipo y personal** (Ver formas) en la tabla de Evaluación del equipo en el renglón de *Calidad del producto* en promedio en las formas de todos los miembros del equipo.

Objetivos como miembro del equipo:

- *Ser un miembro cooperativo y comprometido del equipo.*

Medición posible: En la forma de **Evaluación del equipo y personal** (Ver formas) en la tabla de Evaluación del rol, obtener una puntuación mayor que 3 en *Apoyo y ayuda proporcionada.*
Otra medición posible: En esa forma obtener más de 3 en *Contribución general del rol.*

- *Ser disciplinado*

Medición posible: Haber llenado todas las formas Semana personal en el ciclo 1.
Otra medición posible: Haber cumplido con el Plan personal en un 90%.

- *Entregar los productos de los que es responsable a tiempo*

Medición posible: Haber entregado 95% de los productos en las fechas propuestas.

Referencias bibliográficas

- Braude E. J. *Ingeniería de Software. Una perspectiva orientada a objetos*. Alfaomega 2003.
- Humphrey Watts S., *Introduction to Team Software Process*, SEI Series in Software Engineering, Addison Wesley, 2000.

Capítulo 4

Fase de Estrategia

Objetivos del capítulo:

Explicar los objetivos y las actividades de la fase de Estrategia.

- Cómo plantear y seleccionar una estrategia para dividir el trabajo en dos ciclos de desarrollo.
- Aprender los fundamentos de Administración de la configuración.

4.1 Fase de Estrategia objetivos, actividades y productos.

El objetivo de la fase de Estrategia para el primer ciclo es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos.

En el segundo ciclo, en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Objetivo

O1. Tener una estrategia para dividir el desarrollo en dos ciclos.

A partir del texto donde se define el problema a resolver, se hace una lista de necesidades o funcionalidades que deberá tener el software y se plantean varias estrategias para decidir lo que se hará en cada ciclo de desarrollo. Se explica la técnica de *Gráficas de dependencia* [Oktaba] para la selección de la estrategia de desarrollo en ciclos.

Actividad:

A1.1 Definir y documentar la estrategia

Producto:

Forma Estrategia

Objetivo

O2. Construir un repositorio compartido donde se guardarán los productos de trabajo generados por el equipo.

Actividad:

A2.1 Definir y crear el repositorio.

Producto

Repositorio de productos del equipo

Objetivo

O3. En el segundo ciclo, controlar los cambios a los documentos que se generaron en el primer ciclo.

A3.1 Hacer el Plan de configuración del software.
A3.2 Hacer el informe del estado de la configuración.
A3.3 Hacer el informe semanal del estado de los cambios.

Productos

Plan de configuración.
Forma del Informe del estado de la configuración
Informe semanal del estado de los cambios

Formas a entregar en todas las fases:

- Semana personal
- Semana del equipo
- Minuta de la reunión

Resumen de productos a entregar:

Producto	Responsable
Forma Estrategia	Administrador de desarrollo
Repositorio de productos del equipo	Administrador de apoyo

Resumen de productos a entregar en el segundo ciclo:

Producto	Responsable
Plan de la configuración	Administrador de apoyo
Informe del estado de la configuración	Administrador de apoyo

Solicitudes de cambios	Administrador de apoyo
Informe Semanal del estado de los cambios	Administrador de apoyo

4.2 Definición de la estrategia

La estrategia consiste en imaginar cómo se va a dirigir la construcción del producto. El software se va a construir en dos ciclos, por lo que el equipo debe decidir qué funcionalidades desarrollará en cada uno.

Para establecer la estrategia se debe de hacer varias propuestas alternativas (por lo menos dos) y analizar riesgos y beneficios de cada una y contestar preguntas como las siguientes:
¿Cuáles son las funciones principales del producto?
¿Cómo repartir esas funciones entre los miembros del equipo?

A partir del texto que entrega el instructor donde se define el problema a resolver, se hace la lista de las necesidades o funcionalidades que deberá tener el software. Se plantean varias estrategias de las que se selecciona la más adecuada indicando el alcance de lo que se hará en cada ciclo. Se proponen los siguientes criterios para la selección de la estrategia.

Criterios para la selección de estrategia

1. El producto del primer ciclo debe proporcionar un subconjunto ejecutable de funcionalidades mínimas del producto final.
2. El producto del primer ciclo debe proporcionar una base fácil de extender.
3. El diseño de producto debe tener una estructura modular que permita el trabajo independiente de los miembros del equipo.

Para escoger la estrategia se proponen varias alternativas (por lo menos dos) que se analizan según los criterios comparando los beneficios y desventajas de cada una.

Una técnica para obtener alternativas de estrategias es la *Gráfica de dependencias entre necesidades [Oktaba]*. Esta gráfica permite analizar las funcionalidades o necesidades del problema y sus dependencias de manera gráfica para escoger la mejor.

Gráfica de dependencias entre necesidades

Una *Gráfica de dependencias entre necesidades* es una gráfica dirigida cuyos vértices representan las necesidades o funcionalidades de un problema y los arcos reflejan relaciones entre ellas.

Se lee con cuidado el texto del problema y se identifican las necesidades que se listan y numeran, por ejemplo N1, N2, etc. Se analizan estas necesidades para buscar las dependencias entre ellas. Las relaciones de dependencia pueden ser totales o parciales. A partir de la lista y del análisis de las dependencias se construye la gráfica siguiendo los pasos:

1. Se identifican las necesidades independientes, las que no dependen de otras, y se colocan sus números como nodos terminales de la gráfica. Una necesidad ***a*** se considera independiente si para que se ejecute esa necesidad no es necesario que se haya realizado ninguna otra.
2. Se analizan una por una las demás necesidades, revisando si existen dependencias con las necesidades ya colocadas en la gráfica. Una necesidad ***b*** *depende totalmente* de otra necesidad ***a*** si la satisfacción de ***b*** requiere de la satisfacción completa de la necesidad ***a***. Si este es el caso, la necesidad ***b*** se coloca en la gráfica como un nuevo nodo, unido por un arco continuo dirigido hacia el nodo de la necesidad ***a***.
3. Se repite el paso anterior hasta colocar todas las necesidades y los arcos de dependencias totales en la gráfica.
4. La gráfica resultante se analiza, a fin de identificar los nodos cuyas necesidades se satisfagan parcialmente a partir de la satisfacción de otros nodos. Una necesidad ***b*** *depende parcialmente* de otra necesidad ***a***, si ***b*** puede satisfacerse de modo incompleto mediante la satisfacción completa de la necesidad ***a***. En estos casos, la necesidad ***b*** se conecta mediante un arco dirigido, trazado como una línea discontinua, hacia el nodo de la necesidad ***a***.

Analizando la gráfica resultante se pueden obtener alternativas que cumplen con los criterios para la selección de la estrategia:

1. *El producto del primer ciclo debe de proporcionar un subconjunto ejecutable de funcionalidades mínimas del producto final.*
   La gráfica permite identificar las funcionalidades mínimas para el primer ciclo mediante la selección de los nodos terminales (no necesariamente todos) y una o más trayectorias, de poca longitud, a partir de esos nodos. Los nodos elegidos representan posibles alternativas de necesidades a desarrollar en el primer ciclo.

2. *El producto del primer ciclo debe proporcionar un producto fácil de extender.*
   A partir de los nodos seleccionados como parte de las funcionalidades mínimas, es posible ir añadiendo en el siguiente ciclo otras necesidades de nodos dependientes.
3. *El diseño de producto debe tener una estructura modular que permita el trabajo independiente de los miembros del equipo.*
   La gráfica sugiere una probable división de trabajo. Observando aquellos nodos que tienen más de un arco de entrada (necesidades cuya satisfacción habilita la satisfacción de dos o más necesidades), los nodos descendientes representan posibles desarrollos independientes dentro de la estructura del producto, para que se pueda dividir el trabajo de forma independiente para cada miembro del equipo.

Nota: El análisis se realiza a partir de una descripción en lenguaje natural, lo que puede ser susceptible a distintas interpretaciones. En consecuencia, es de esperar que este análisis pueda generar distintas gráficas de dependencias para la misma descripción del problema.

Un ejemplo de cómo construir esta gráfica a partir del siguiente enunciado de problema es el siguiente:

Necesidades de un sistema procesador de encuestas con varios temas

Objetivo del problema: Desarrollar un sistema que apoye a un usuario que desea realizar encuestas a una población dada sobre algunos temas de interés.

La lista de necesidades y su nombrado son las siguientes:

N1. El sistema pedirá al encuestado los datos personales y le presentará las preguntas de la encuesta con sus opciones.

N2. El usuario podrá definir el tema de la encuesta, los datos personales de los encuestados que le interesan, y las preguntas de la encuesta, así como las posibles opciones de respuesta.

N3. El sistema podrá efectuar algunas estadísticas directas sobre el tema de la encuesta, como cuántas personas eligieron cada opción de cada pregunta y qué porcentaje son de la población total.

N4. El sistema permitirá imprimir las estadísticas de las encuestas por tema.

N5. El sistema permitirá imprimir una encuesta vacía.

N6. El sistema permitirá imprimir cada encuesta contestada.

N7. El usuario podrá definir varios temas para las encuestas con sus preguntas correspondientes.

N8.El sistema permitirá imprimir las estadísticas de varios temas de las encuestas.

N9. Se programará en Java y se usarán tablas para guardar los datos.

N10. La presentación del sistema podrá ser (a) gráfica (usando GUI) o tipo (b) texto.

En la figura 4.1 se muestra la gráfica de dependencias para estas necesidades. Ejemplos de alternativas de estrategias son:

Alternativa 1.
Ciclo 1 Definir la encuesta aplicarla e imprimirla: N2, N1, N5, N6, N9, N10a
Definir la encuesta (N2)
Imprimirla vacía (N5)

El encuestado la responde (N1)
Se imprime la encuesta llena (N6)
El sistema está hecho en Java y con tablas (N9)
El sistema tiene una interfaz gráfica (N10a)

Ciclo 2 El resto de los nodos

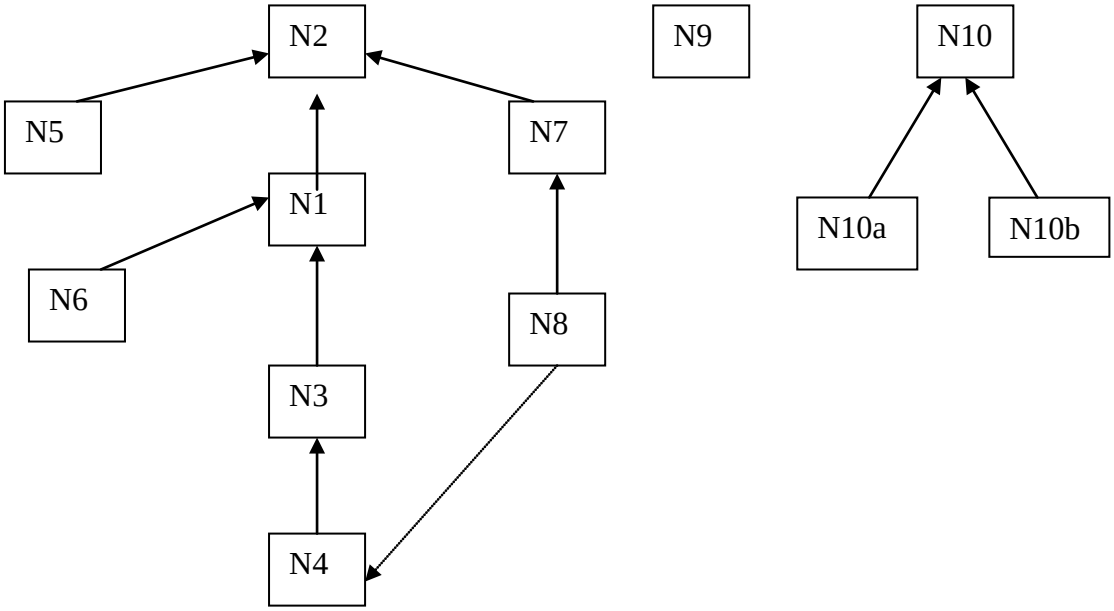


Figura 4.1 Gráfica de dependencias

Alternativa 2.

Ciclo 1 Definir la encuesta, aplicarla y sacar estadísticas: N2, N5, N1, N3, N4, N9, N10a
Definir la encuesta (N2)
Imprimirla vacía (N5)
El encuestado la responde (N1)
Se pueden hacer estadísticas sobre las respuestas de la encuesta (N3)
Se imprimen las estadísticas (N4)
El sistema está hecho en Java y con tablas (N9)
El sistema tiene una interfaz gráfica (N10a)

Ciclo 2 El resto de los nodos

Alternativa 3.

Ciclo 1 Definir varios tipos de encuestas e imprimirlas: N2, N5, N7, N9, N10a,
Definir la encuesta (N2)
Imprimirla vacía (N5)
Se pueden definir varias encuestas sobre diversos temas (N7)
El sistema está hecho en Java y con tablas (N9)
El sistema tiene una interfaz gráfica (N10a)

Ciclo 2 El resto de los nodos

Alternativa 4.
Ciclo 1 Definir la encuesta, aplicarla y sacar estadísticas: N2, N1, N3, N4, N9, N10b
Definir la encuesta (N2)
El encuestado la responde (N1)
Se pueden hacer estadísticas sobre las respuestas de la encuesta (N3)
Se imprimen las estadísticas (N4)
El sistema está hecho en Java y con tablas (N9)
El sistema tiene una interfaz modo de texto (N10b)
Ciclo 2 El resto de los nodos

Cada equipo discute las alternativas y selecciona una justificando su elección en la forma Estrategia.

Forma Estrategia

Esta forma tiene el objetivo de documentar la estrategia seleccionada indicando qué funcionalidades se desarrollarán en cada. En el llenado de esta forma el responsable es el administrador de desarrollo y participa todo el equipo.

La forma tiene un encabezado en que se pone el nombre del equipo y ciclo. En el primer recuadro se dibuja la gráfica de dependencias según la técnica explicada. La justificación de la estrategia elegida en el segundo recuadro. En la tercera tabla se indican las funcionalidades que se implementarán en cada ciclo, marcando con un X la columna correspondiente.

Funcionalidades o necesidades	**Ciclo 1**	**Ciclo 2**

4.3 Definir el repositorio del equipo

Para facilitar el acceso a todos los componentes que se van generando, en el primer ciclo, se crea un repositorio electrónico que deberá ser accesible a todos los miembros del equipo.

El repositorio deberá tener los siguientes elementos:
Portada – nombre del equipo, nombres de los miembros y sus roles, fechas de inicio y fin del proyecto.
Una carpeta para cada ciclo con las carpetas para cada fase.
Una carpeta para los documentos comunes del equipo como: minutas de las reuniones, formas Semana del equipo, formas de Registro de defectos, estándares, etc.

Ejemplo de la estructura del repositorio:

Equipo Alfa
- Nombres y roles
- Estándares
- Ciclo 1
  - Lanzamiento
  - Estrategia
  - … (las demás fases)
  - Documentos
    - Minutas
    - Formas Semana del equipo
    - … (las demás formas del equipo)
- Ciclo 2
  - …

Para guardar los documentos en el repositorio se define un estándar de nombrado de los archivos para que todos los miembros del equipo los nombren de manera similar y los puedan identificar fácilmente. Generalmente se usan acrónimos que incluyan el nombre del documento, el ciclo, fase y versión.

4.4 Administración de Configuración de Software

La *Administración de Configuración de software* es un conjunto de actividades que se llevan a cabo para tener el control sobre los cambios y las versiones del producto. Una configuración de software es la colección consistente de software y su documentación que se entrega a un cliente. Los objetivos de la administración de configuración son: asegurar que se conoce la versión aprobada de cada elemento de la configuración, que a todos los elementos aprobados se controlen los cambios. Todo esto para que la integración de una configuración de software se haga con las últimas versiones aprobadas tanto del código como de los documentos y que quede registro de las razones de los cambios para fines de futuro mantenimiento.

Las actividades de la administración de la configuración son:

1. Hacer el Plan de administración de la configuración.
   - Definir los elementos de la configuración.
   - Definir mecanismos de control los cambios.
2. Hacer un informe del estado de la configuración.
3. Auditar la configuración.

En el primer ciclo se están generando varios documentos que se modificarán en el segundo, por lo que se debe llevar un buen control de cambios y de versiones. El responsable de llevar la administración de la configuración es el Administrador de apoyo.

- **Plan de configuración**

Para llevar a cabo las actividades de la administración de la configuración se debe generar un plan. Es un documento en el que se indica cuáles son los productos de trabajo, que formarán parte de la configuración, a los que se controlarán los cambios. El plan también debe de definir cómo se controlarán los cambios.

En el Plan de configuración el Administrador de apoyo indica:

- Los productos de trabajo del proyecto a los que se controlarán los cambios.
- Establecer la autoridad que autorizará los cambios a la que se llama *comité de control de cambios.*

- **Definir los componentes.**

Los componentes son los documentos y productos que se han generando en el proceso de desarrollo. Se define quién es el responsable de cada componente y que será el encargado de efectuar los cambios y probarlos.

Los componentes y sus responsables son:

Fase	**Componente**	**Responsable**
Lanzamiento	Objetivos	Líder del equipo
	Estándar de documentación	Administrador de calidad
	Registro de riesgos	Administrador de planeación
Estrategia	Forma Estrategia	Administrador de desarrollo
	Plan de la configuración	Administrador de apoyo
Planeación	Plan del equipo	Administrador de planeación
	Plan de calidad	Administrador de calidad
Captura de requerimientos	Diagrama de casos de uso	Administrador de desarrollo
	Detalle de los casos de uso	
	Prototipo	
	Requerimientos no funcionales	
	Glosario	
	Plan de prueba del sistema	
Diseño	Estándar del diseño	Administrador de desarrollo
	Diagrama de paquetes	
	Diagrama de distribución	
	Diagrama de clases	
	Diagramas de secuencia	
	Diagrama de estados	
	Plan de pruebas de integración	
Implementación y pruebas unitarias	Diagramas de clases detallados	Administrador de desarrollo
	Plan de pruebas unitarias	Ingenieros de desarrollo
Prueba del	Informe de las pruebas del sistema	Administrador de desarrollo

sistema	Manuales de usuario, de instalación	
Cierre	Evaluación de equipo y personal	Líder del equipo
	Informe de mediciones	Administradores de calidad y planeación
	Lecciones aprendidas y sugerencias de mejora	
	Informe del estado de la configuración	Administrador de apoyo

- **Control de cambios.**

Un concepto importante de la administración de la configuración es el de *línea base*. Se considera que un componente está en *línea base,* si ya ha sido aprobado y ha sido guardado en el depósito del proyecto como aprobado. La línea base del proyecto son todos los componentes ya revisados y aprobados en el primer ciclo.

Si se le quiere hacer algún cambio deberá solicitarlo al comité de control de cambios, a través de la forma de Solicitud de cambios.

La autoridad que aprobará las solicitudes, es el *comité de control de cambios*, generalmente está formado por el administrador de apoyo y el administrador de desarrollo para que no sea muy burocrático.

Cualquier miembro del equipo puede solicitar un cambio a un componente en línea base. Un cambio puede deberse a que se encontró un defecto en el producto y deberá corregirse o se ampliarán su alcance.

Para solicitar un cambio, se llena la solicitud de cambio y se entrega al comité de control de cambios para su análisis. Para cada componente que se quiera cambiar se llena una forma.

Si el comité de control de cambios aprueba el cambio, el administrador de apoyo proporciona una copia al responsable del componente para que haga el cambio, lo prueba y se lo entrega al administrador de calidad para su revisión. El administrador de calidad lo revisa y se lo entrega al de apoyo para que lo incorpore en la línea base como nueva versión. Si no se aprueba, el comité documenta las razones del rechazo.

Forma de Solicitud de cambios

Se solicitan los cambios a través de la *Forma de Solicitud de cambios* y el administrador de apoyo debe llevar el control de esas formas.

La forma de Solicitud de cambios tiene un encabezado semejante a todas las formas del equipo. La información que se debe documentar es:
Información del componente. Nombre, responsable y dirección donde está el componente.
Razones para hacer el cambio. Se describe el defecto encontrado o lo que ocasiona la solicitud del cambio.

Impactos del cambio. A qué otros componentes afectará el cambio propuesto.
Descripción del cambio. Se describe los cambios que se le harán al componente.
Estado del cambio. Cuando el comité de control de cambios decide se indica si el cambio fue aceptado, rechazado o está en proceso de revisión.

Resumiendo, las tareas a realizar para el control de cambios son:
- El administrador de apoyo recibe las solicitudes de cambios.
- Pasa la solicitud al comité de control de cambios para que lo apruebe.
- Si se aprueba, entregar una copia del componente al responsable para que se haga el cambio y se pruebe.
- El administrador de calidad revisa que el cambio y los componentes afectados.
- El administrador de apoyo recibe los componentes aprobados los resguarda en el depósito.
- Al terminar el ciclo se construye la configuración apropiada del sistema, usando las versiones aprobadas de código y documentos.

- **Informe del estado de la configuración**

En el segundo ciclo el administrador de apoyo hace un *Informe del estado de la configuración* por medio de la forma correspondiente a fin de que el equipo sepa cuáles componentes se generaron, en qué versión están y en qué dirección están almacenados.
Esta forma tiene un encabezado semejante a las otras formas del equipo.

En la primera tabla se listan todos los documentos ya generados en el primer ciclo, las versiones finales y su dirección en el depósito. Esta información deben conocerla todos los miembros del equipo para que utilicen las versiones aprobadas como base para el desarrollo del segundo ciclo.

Elementos de la configuración al final del ciclo

Elemento de la configuración	Versión	Dirección donde se encuentra

- **Auditoria de la configuración**

La actividad *Auditoria de la configuración*, tiene por objetivo dar a conocer la información sobre los cambios que se solicitan. Se deberá saber cuántos cambios se solicitan, cuáles son aprobado por el comité de control de cambios, cuáles rechazados y cuáles habían sido aprobados pero se requiere de deshacerlos.

Para informar al equipo del estado de los cambios cada semana, el administrador de apoyo proporciona la *Forma Semanal del estado de los cambios* que estará disponible para la consulta de todo el equipo para saber el estado de los cambios propuestos.

Esta forma tiene un encabezado igual a todas las formas del equipo y consiste de 2 tablas que son:

Informe de la actividad de control de cambios

Formas de Solicitud de cambios:	**En esta semana**
Enviadas	
Aprobadas	
En proceso	
Rechazadas	
En reversa (cambio que debe deshacerse)	

En esta tabla se informa del estado de las solicitudes de cambio en la semana, cuántas se enviaron, si están aprobadas y están en proceso, rechazadas o revertido el cambio.

La segunda tabla hace un informe de cada elemento de la configuración sometidos en esta semana para darles un seguimiento.

Esta tabla es muy útil para que el equipo sepa con que elementos trabajar y no lo haga con versiones obsoletas.

Estado de los cambios en esta semana:

Elemento de la configuración	**Versión**	**Estado de los cambios propuestos**	**Estado de la implementación**

4.4 Profundización de los temas para el segundo ciclo (revisar)

4.4.1 Medidas del tamaño del software

Para medir el tamaño de un software se podría hacer en términos de los bytes que ocupa, o del número de líneas de código que tiene o de la funcionalidad que proporciona, los reportes que genera, las pantallas que presenta, el número de tablas con que cuenta la base de datos, etc. [Durán]

Pero como se habrá comprobado, un software no solo es el código, sino todas las actividades que se debieron hacer para generar el código: la planeación, el modelado de casos de uso, el diseño, planeación de pruebas a todos los niveles, etc. La práctica muestra que para establecer el tamaño de software es mejor considerar los requerimientos del software, esto es, las funcionalidades que tendrá. Así que, basados en la experiencia

obtenida se asigna un esfuerzo a cada funcionalidad lo que servirá de base a la estimación del tamaño. Una técnica que usa este principio es la siguiente:

- Puntos de función (Function Points) que miden la complejidad y eficiencia desde el punto de vista del usuario. Establecido por el International Function Point Users Group. Estas estimaciones son independientes del lenguaje, se expresan en términos de funcionalidades del sistema, sus entradas, salidas, comunicaciones con otros sistemas, complejidad de operaciones y cualidades de sistema.

Para mayor información sobre esta técnica consultar http://www.functionpoints.com/

Referencias bibliográficas

- Durán S. “Métricas de tamaño de software basadas en funcionalidad”. Software Guru, mayo-junio 2005. p.30.
- Oktaba H., Ortega A. Jorge L. “Una propuesta de análisis de necesidades mediante las gráficas de dependencia en la fase de estrategia de TSPi “. Memorias de IDEAS 2002, La Habana, Cuba 2002.
- Humphrey Watts S., *Introduction to Team Software Process*, SEI Series in Software Engineering, Addison Wesley, 2000.
- Pressman R. S. *Ingeniería de Software*. Mc Graw Hill 3a edición 1992.

Capítulo 5

Fase de Planeación

Objetivos del capítulo:

Los objetivos de este capítulo son dar los elementos para el proceso de la fase de Planeación.

- Explicar cómo se hacen el Plan de trabajo del equipo.
- Introducir las revisiones entre colegas.

5.1 Fase de Planeación (ciclo 1) objetivos, actividades y productos.

El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién.

En esta misma fase se introduce el concepto de calidad y el de las *revisiones entre colegas* como técnica para fomentar la calidad, la cual tendrá que aplicarse a lo largo del desarrollo.

Objetivo

O1 Planear el trabajo del equipo.

Planear el trabajo del equipo para el primer ciclo, incluye hacer la lista de las actividades a realizar en el proceso de desarrollo.

Actividades:

A1.1 Identificar las actividades a realizar en el desarrollo.

Para planear lo que se debe hacer cada semana se consideran las actividades propias del desarrollo, las actividades propias de cada rol, las que todos deben realizar todas las semanas y las de calidad.

A1.2 Diagramar la planeación del equipo con un diagrama de Gantt.

Para ver gráficamente la planeación se hace un diagrama de Gantt, en el que se representan las actividades que se pueden hacer en paralelo, cuáles son dependientes de la terminación de otras, la duración estimada de cada una y los responsables.

Productos

Plan del equipo.

Objetivo

O2. Incluir actividades de verificación de los productos.

Dentro del proceso de desarrollo, se realizan actividades para verificar la calidad de los productos. Una técnica de verificación es la de *revisión entre colegas* que ayuda a detectar defectos desde fases tempranas del producto y corregirlos antes de que ocasionen mayores problemas.

Actividad

A2.1 Hacer las reuniones de revisión entre colegas.

El objetivo de las reuniones de revisión entre colegas, es buscar defectos en los productos generados esa semana. Los defectos encontrados se registran en la forma de *Registro de defectos.*

Productos

Lista de verificación.

Forma de Registro de defectos.

Formas a entregar a partir de esta fase y en todas las fases:

- Semana personal
- Semana del equipo
- Mminuta de la reunión
- Lista de verificación
- Registro de defectos

Resumen de productos a entregar:

Producto	Responsable
Plan del equipo	Administrador de planeación
Lista de verificación	Administrador de calidad
Forma de Registro de defectos	Administrador de calidad

5.2 Planeación

Una técnica fundamental de un proyecto exitoso, es planear el trabajo y darle seguimiento a lo largo del proyecto.

Planear sirve para:

- que cada quien sepa qué debe hacer y cuándo.
- permite considerar tareas que se podrían olvidar.
- ayuda a cumplir los compromisos.
- permite dar seguimiento al plan del proyecto y adecuarlo.

Una técnica para identificar y describir las actividades que se tienen que realizarse en un proyecto es *La Estructura de Descomposición del Trabajo (Work Breakdown Structure WBS*). Esta técnica consiste en hacer un árbol que denota las actividades a realizar partiendo de las más generales, que a su vez se descomponen en más precisas y así sucesivamente hasta llegar a tareas atómicas. Los elementos atómicos del WBS son paquetes de trabajo o actividades detalladas que se incluyen en el plan.

Para construir buenos diagramas de WBS se pueden seguir estos criterios: [Cantor p.140]

- Cada tarea atómica es precisa, esto es, se sabe como hacerla y tiene bien definida su duración.
- Pueda rastrearse el estado de la tarea.
- Las tareas tienen un buen grado de granularidad, ni muy generales ni demasiado detalladas.

La estructura de las actividades se representa por medio de una jerarquía. En la raíz se pone el proyecto completo. En el siguiente nivel los tipos de actividades del proyecto. En el siguiente nivel el desglose en tareas. Se puede numerar las actividades según su nivel con una nomenclatura como sigue: la raíz se numera como 1, las actividades del segundo nivel como 1.1, 1.2, etc. En el tercer nivel como 1.1.1, 1.1.2, o 1.2.1, etc.

Actividades del Proceso de desarrollo

Aplicando esta técnica para la identificación de las actividades a realizar en el curso, se establecen cuatro tipos de actividades:

1. De desarrollo, según la fase en que se está
2. De los roles
3. Las que todos deben hacer todas las semanas.
4. Que fomentan la calidad.

Para hacer la planeación es necesario hacer la lista de las tareas a realizar en cada tipo de actividades y estimar el tiempo que se lleven.

Algunas fases del desarrollo se llevan dos semanas en realizarse, estas fases son:

- Especificación de requerimientos
- Diseño
- Construcción

Otras fases se hacen en una semana y son:

- Integración y prueba del sistema
- Cierre

Para aplicar la técnica de WBS, se numeran estos cuatro tipos de actividades a partir de la actividad principal del proyecto que es el desarrollo de software, quedando como siguen:

1. Proyecto de desarrollo de software
    1.1 Actividades de desarrollo

1.2 Actividades de los roles
1.3 Actividades de todos, todas las semanas
1.4 Actividades de calidad

Las actividades de desarrollo son:

1.1 Actividades de desarrollo

Las tareas de este tipo de actividades se dividen por semana y todos participan en ellas.

1.1.1 Especificación de requerimientos

Primera semana de Especificación de requerimientos:
1.1.1.1 Entender cuáles son las necesidades o requerimientos del software
1.1.1.2 Construir el glosario
1.1.1.3 Especificar los requerimientos funcionales con casos de uso
1.1.1.4 Detallar los casos de uso

Segunda semana de Especificación de requerimientos:
1.1.1.5 Hacer el prototipo de la interfaz de usuario
1.1.1.6 Definir los requerimientos no funcionales
1.1.1.7 Hacer el Plan de pruebas del software

1.1.2 Diseño

Primera semana del Diseño:
1.1.2.1 Entender los principios del diseño para trabajo en equipo.
1.1.2.2 Decidir las tecnologías y el ambiente de implementación
1.1.2.3 Establecer el estándar de diseño.
1.1.2.4 Plantear la arquitectura con un diagrama de paquetes
1.1.2.5 Hacer el diagrama de distribución del sistema
1.1.2.6 Hacer los diagramas de clases para los paquetes de la arquitectura

Segunda semana del Diseño:
1.1.2.7 Hacer los diagramas de secuencia para los casos de uso normales y con situaciones excepcionales
1.1.2.8 Modelar por medio de diagramas de estado la navegación en la interfaz de usuario o la secuencia válida de casos de uso en el sistema
1.1.2.9 Hacer el Plan de pruebas de integración

1.1.3 Construcción

Primera semana de Construcción
1.1.3.1 Producir el código de los componentes
1.1.3.2 Planear y hacer las pruebas unitarias

Segunda semana de Construcción
1.1.3.3 Hacer la integración personal de los componentes

1.1.4 Integración y prueba del sistema

1.1.4.1 Hacer la especificación de la integración
1.1.4.2 Aplicar las pruebas de integración siguiendo la especificación
1.1.4.3 Aplicar las pruebas del software completo siguiendo el Plan de pruebas de software
1.1.4.4 Hacer los manuales

1.1.5 Cierre
1.1.5.1 Entrega de la primera versión del producto funcionando
1.1.5.2 Evaluar el cumplimiento de los objetivos personales.
1.1.5.3 Evaluar el desempeño del equipo y la asignación de roles.
1.1.5.4 Identificar las lecciones aprendidas y propuestas de mejora

En el segundo tipo de actividades están las que se deben realizar según el rol que está jugando cada miembro del equipo.

1.2 Actividades de los roles (Revisar los capítulos de cada rol, se ejemplifica con las actividades generales a realizar cada semana)

Líder:
1.2.1 Efectuar la reunión del equipo y prepara la agenda

Administrador de desarrollo:
1.2.2 Dirigir al equipo en las actividades de desarrollo en cada una de las fases

Administrador de planeación:
1.2.3 Recolectar las formas semana personales y hacer la semana del equipo

Administrador de calidad
1.2.4 Efectuar la reunión de revisión

Administrador de apoyo:
1.2.5 Conformar las herramientas para el desarrollo en el primer ciclo y llevar el control de cambios en el segundo

El tercer tipo de actividades, son las que todos deben realizar, son mas cortas y se realizan en paralelo a los otros tipos de actividades.

1.3 Las actividades de todos, todas las semanas.
1.3.1 Asistir a la reunión del equipo.
1.3.2 Llenar y entregar la forma Semana personal.
1.3.3 Hacer las correcciones indicadas por el instructor a los productos de las que es responsable.

El cuarto tipo, son las actividades de calidad, se realizan cuando las de desarrollo han terminado y antes de entregar los productos al instructor.

1.4 Actividades de calidad

1.4.1 Asistir a las reuniones de revisión entre colegas.

Teniendo la lista de las actividades, se revisa el Guión general del curso que indica en que semana se deben realizar. Para hacer el Plan del proyecto se identifican las que se pueden realizar en paralelo. Se representa gráficamente el plan con la técnica de Diagrama de Gantt.

Diagramas de Gantt

Los diagramas de Gantt se usan en la planeación de los proyectos. Muestran gráficamente las actividades y su distribución en el tiempo. Permiten dar seguimiento a las actividades en la realización de un proyecto.

Un diagrama de Gantt es una tabla en que las columnas representan el tiempo del proyecto, generalmente una columna para cada día agrupados en semanas. En los renglones se ponen las actividades a realizar. Una línea horizontal en un renglón muestra el periodo de tiempo en que se realizará una actividad desde el día en que se inicia hasta su terminación. Cada actividad puede tener un responsable de que se lleve a cabo que se establece en una columna de recurso (donde el recurso es la persona)

Este diagrama puede representar dependencias entre las actividades. Una actividad es dependiente de otra, si se debe iniciar hasta que se terminó la primera actividad. En los diagramas esto se representa por líneas que se inician cuando la otra actividad termina. Las actividades paralelas son las que se pueden hacer al mismo tiempo. Se muestra con líneas que ocupan los mismos días para renglones (actividades) diferentes.
Hay varias herramientas para hacer estos diagramas.

Pasos para construir un diagrama de Gantt:

1. Hacer la lista de las actividades a realizar en el ciclo y ponerlas en los renglones. Para identificar estas actividades se usa la técnica de WBS explicada anteriormente.
2. Identificar las semanas que durará el ciclo y poner los días y semana en las columnas.
3. Para cada actividad/renglón estimar el tiempo que se llevará en realizarlo marcando los días/columnas. Identificar las actividades paralelas y las dependientes.
4. Asignar responsables de cada actividad.
5. Revisar y ajustar el diagrama.

Proceso de planeación para el primer ciclo.

El administrador de planeación guía al equipo en la planeación con auxilio de un diagrama de Gantt.

1. Todo el equipo revisa la lista de las tareas a realizar en el ciclo según la técnica de WBS y la lista anterior que incluye todos los tipos de actividades a realizar en el proyecto.

2. Se identifican las semanas correspondientes al primer ciclo. En el guión del curso se tienen las fases y las fechas del curso.
3. Se pone la lista en los renglones y una línea para cada tarea en la semana en que se realizará indicando el día que deberá iniciarse y terminarse. Se identifican tareas dependientes y paralelas. Esto se muestra en el diagrama de Gantt, en líneas paralelas para actividades que pueden ser simultáneas. En las tareas dependientes la fecha de inicio será al terminar la actividad de la que depende. Se asignan responsables de cada actividad.
4. Revisar esta planeación hasta que todos estén conformes.
5. Cada semana el administrador de planeación, resume en la forma Semana del equipo, las actividades realizadas y las compara con la planeación que se hizo en el diagrama de Gantt. Si hay retrasos les comunica al equipo, para tomar las acciones correctivas. Si están en tiempo, se continúa así. Si hay ganancia de tiempo se comunica al equipo para disponer de él según se ofrezca.

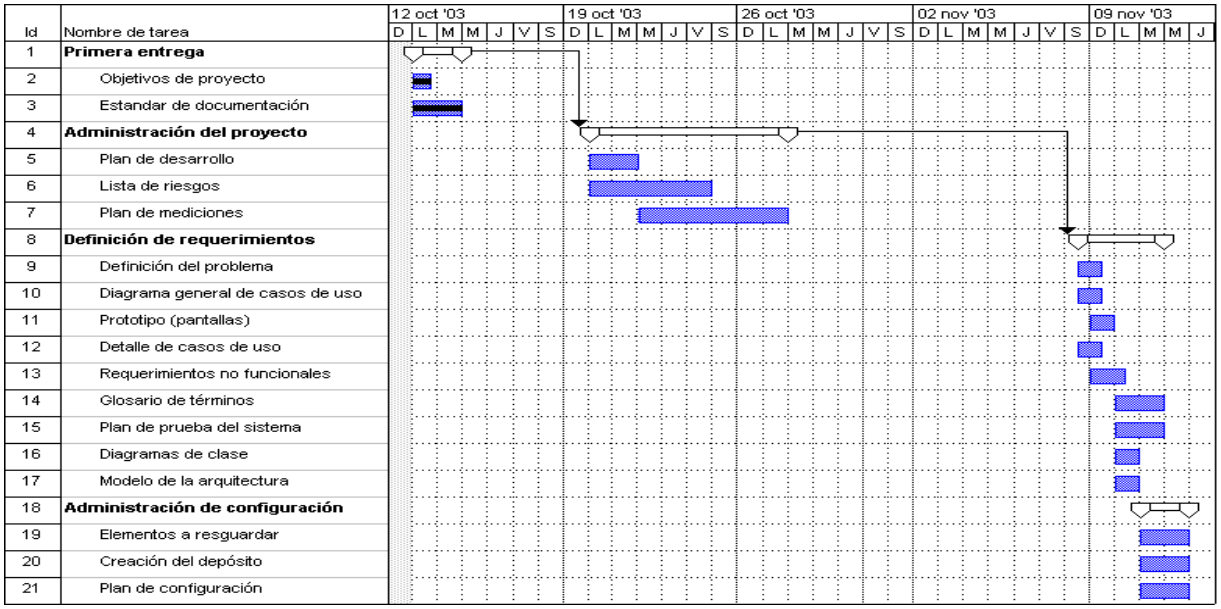


Figura 5.1 Ejemplo de un diagrama de Gantt

5.3 Revisiones entre colegas

Calidad de un producto de software es el grado en que éste satisface los requerimientos especificados, así como las necesidades y expectativas del usuario. Un objetivo importante del curso es aprender a construir software de *calidad*. Para obtener calidad es necesario que todos los productos que se generen sean consistentes y no tengan defectos. El problema que tenemos los humanos es que nos equivocamos y, sin querer, introducimos defectos en lo

que producimos. Un *defecto* es resultado de un error cometido por un desarrollador al generar un producto. Los defectos aún pequeños como faltas de ortografía o de dedo, pueden ocasionar problemas severos en el software al presentar inconsistencias o respuestas impredecibles. [Humphrey 1997 p. 4]

Generar productos de calidad implica realizar actividades para asegurarse que no hay defectos. Dos técnicas que se usan para comprobar la calidad de los productos son la *verificación* y *validación*. Verificar un producto tiene por objetivo revisar que no tenga defectos. Validar el software es asegurarse que hace lo que debe hacer. Para validar se usa la prueba del software de la que se hablará en los capítulos 9 y 10.

Está comprobado que cuando un autor detecta un defecto en su producto, lo puede corregir. Para ayudar a los autores de los productos a encontrar defectos, una técnica muy útil, es pedir a un colega que le ayude a buscarlos. Esta técnica de verificación tiene los nombres formales de r*evisiones entre colegas* o de *inspección entre pares* (*peer reviews* en inglés).

Las revisiones entre colegas consisten en solicitar a un colega, que tenga un conocimiento parecido al nuestro, que revise nuestro producto y busque defectos. Dentro del proceso de desarrollo se introduce esta técnica a fin de mantener los productos lo más limpio de defectos que se pueda desde el inicio.

Experimentos han demostrado que las revisiones entre colegas permiten encontrar la mayoría de los defectos, que la mayoría de defectos que normalmente se encuentran durante la fase de pruebas. Sin embargo, pero son mucho más baratas, ya que el costo de encontrar y corregir un defecto en una fase posterior del desarrollo es 10 veces mayor que en una fase anterior.

Otra ventaja de las revisiones entre colegas, es que al revisar el trabajo de los demás, se gana conocimiento de lo que se está haciendo en conjunto. Esto tiene varias ventajas: todos conocen el trabajo de los demás, nadie es indispensable por ser el único que conoce un producto y la comunicación en el equipo se incrementa disminuyendo las inconsistencias en los productos. Un riesgo que se debe evitar en esta técnica, es que se hagan críticas personales al trabajo de los demás. Hay que entender muy bien que el objetivo es mejorar los productos por el bien del proyecto y no hacer críticas al productor.

Para que la revisión sea exitosa y se encuentre el mayor número de defectos es útil saber dónde pueden estar los defectos. Para tal efecto el administrador de calidad prepara una *lista de verificación* para el producto en cuestión que contiene las fuentes más comunes de defectos. El que revisa el producto usa esta lista como guía de apoyo. Los defectos encontrados de registran en la forma de Registro de defectos.

Productos de las revisiones entre colegas

- **Lista de verificación para cada producto a revisar (Checklist)**

El administrador de calidad prepara una lista de verificación indicando dónde podría haber un defecto u omisión. Es más fácil encontrar defectos si se sabe que se está buscando en cada producto.

En todas las listas de verificación se debe incluir que el producto tenga:

- Nombre del producto
- Fecha
- Versión
- Índice
- Portada
- Paginación
- No haya faltas de ortografía, etc.

Además, los posibles defectos u omisiones específicos para el producto en revisión. Por ejemplo, en el documento de Especificación de Requerimientos se deben incluir todos los casos de uso para el ciclo, los nombres de los actores deben ser consistentes, que exista al menos un caso de uso por actor, etc.

- **Forma de Registro de defectos**

Al hacer las revisiones entre colegas se van registrando los defectos que se encuentran. Para llevar una historia de ellos se usa la forma Registro de defectos. Cada revisor tiene una forma y la lista de verificación para saber que está buscando.

LISTA DE VERIFICACION:

Los documentos deben contener:

- Portada
- Índice
- Encabezado con los siguientes elementos:
  - Logo
  - Nombre del proyecto
  - Nombre del documento
  - Clave
  - Versión
  - No. de pagina
- Pie de página en la última hoja con los siguientes elementos:
  - Nombre y puesto de quién lo elaboró
  - Nombre y puesto de quién lo revisó
  - Nombre y puesto del líder del equipo que lo autorizó
  - Fecha
- Ortografía correcta
- Los puntos requeridos en el documento, es decir que este completo.

En las minutas se revisará:

- Encabezado con los siguientes elementos
  - Logo
  - Nombre del proyecto

Figura 5.2 Ejemplo de una lista de verificación

La forma Registro de defectos tiene un encabezado, que identifica el equipo y el ciclo, y contiene una tabla para cada producto revisado en la que se describen los defectos encontrados y el total de defectos por producto.

Producto: **Fecha:**
Descripción de los defectos:

1.
2.
3.

Finalmente se totaliza cuántos productos se revisaron y cuántos defectos se encontraron en esa reunión de revisión.

- **Producto revisado y corregido.**

Una vez detectados los defectos en un producto, el administrador de calidad lo regresa al responsable para que los corrija. Cuando esto suceda, se actualiza la forma Registro de defectos indicando que el defecto fue corregido. Una vez corregidos todos los defectos el producto está listo para su entrega.

Calendario de las reuniones de revisión entre colegas

La reunión de revisión entre colegas se debe de realizar antes de entregar los productos al instructor, con la anticipación suficiente que permita la corrección de defectos.En la reunión participan el administrador de calidad y el miembro del equipo que fue designado para esa fecha. No se recomienda que participe el responsable del producto.

El Administrador de calidad establece las fechas y horario de las revisiones entre colegas y acuerda con los miembros del equipo en que semana participarán como revisores. En todas las reuniones participa el administrador de calidad que las coordina, pero cada semana algún miembro del equipo participa en ellas. El calendario de reuniones de revisión entre colegas es una tabla con las fechas de las reuniones (por lo menos una por semana) y quién quedo designado como participante.

Proceso para las reuniones de revisión.

El administrador de calidad deberá:

1. Revisar el calendario de revisiones y convocar al participante.
2. Hacer la lista de verificación de los productos a revisar.
3. Dirigir la reunión de revisión entre colegas.
   Se revisan los productos.
   Se registran los defectos en la forma.

4. Regresar los productos con el registro de defectos a los responsables para que los corrijan.
5. Completar la forma indicando cuántos defectos se encontraron, en qué producto, el tiempo que duró la revisión.
6. Registrar la corrección de defectos en la forma.

Equipo de revisión

1. Revisar de forma individual el producto apoyándose en la lista de verificación.
2. Registrar los defectos en la forma.

Responsable del producto

1. Discutir y analizar los defectos encontrados en la revisión.
2. Corregir defectos.
3. Entregar el producto corregido al administrador de calidad.

5.4 Profundización de temas para el segundo ciclo

5.4.1 Administración de Proyectos

En Ingeniería de Software un *proyecto* se puede definir como un esfuerzo organizacional para desarrollar, introducir o evaluar una aplicación nueva, una herramienta o un método. Es un esfuerzo durante un periodo de tiempo delimitado, con un equipo de trabajo asignado y con recursos disponibles para producir un cierto resultado. En caso del desarrollo, el resultado a obtener es un producto que es un sistema de software para un cliente distinto del equipo de desarrollo.

La *Administración de Proyectos* es una función auxiliar para apoyar el buen desarrollo de los proyectos. Sus actividades principales son: Planeación del Proyecto y el Seguimiento del Proyecto hasta su cierre.

Planeación del Proyecto

La planeación del proyecto parte de la definición y entendimiento de los objetivos del proyecto y de los criterios de éxito (por ejemplo, aceptación y pago del cliente) para realizar la estimación de los recursos requeridos y su calendarización.

La estimación del costo del proyecto se basa en la estimación del esfuerzo para realizar el proyecto. El esfuerzo se mide en horas (semana o mes)/hombre de recursos humanos requeridos para su desarrollo. Para estimar el esfuerzo se necesita saber que tipo de actividades se tienen que desarrollar para lograr los objetivos del proyecto.

Otro elemento necesario es la estimación de la productividad de los recursos humanos medida en términos de tamaño del producto entre el tiempo dedicado por una persona para

producirlo. La típica medición de la productividad es miles de líneas de código (KLOC) entre persona por año (person-year), abreviado como KLOC/PY.

La productividad medida de esta manera no se puede comparar fácilmente entre organizaciones porqué pueden existir diferencias en las mediciones del tamaño y del tiempo. Por ejemplo, en el caso de tiempo puede haber diferencia por:

- Las actividades que se cuentan. Por ejemplo, si se incluye especificación de requerimientos, análisis, diseño y pruebas o se cuenta solo la construcción.
- El personal que se incluye. Por ejemplo, solo se considera al programador o también el tiempo del resto del personal de equipo.
- El tiempo que se cuenta, solo el productivo o también el improductivo.

En el caso de medición de líneas de código (LOC) también puede haber diferencias en qué líneas cuentan:

- sólo con comandos ejecutables o también líneas con declaraciones
- líneas físicas o líneas lógicas
- lenguaje de máquina o lenguaje de alto nivel
- de nuevo desarrollo, reusadas y cambiadas.

El enfoque más razonable es contar solo las líneas nuevas, lógicas, en un lenguaje de alto nivel, excluyendo comentarios, contando comandos y declaraciones.

Se recomienda que dentro de la misma organización se use la misma definición para medir el tamaño de los productos de software en todos los proyectos para tener una medida consistente de productividad.

La medición de tamaño en LOC tiene el inconveniente de que es dependiente del lenguaje de programación. Si en un una organización se usan varios lenguajes hay que medir la productividad por cada lenguaje, esto puede complicar la recolección de los datos y su uso.

Existe otra forma de medir el tamaño de productos de software independiente de los lenguajes de programación, conocida como Puntos de Función (*Function Points FP)* propuesta por Albrecht. En este caso, lo que se cuenta son diferentes elementos de funcionalidad del sistema tales como: entradas, salidas, archivos, tablas y consultas. Tales elementos fueron calibrados y se les asignan pesos según las tablas con datos históricos de muchos proyectos desarrollados en el mundo. Las tablas iniciales se elaboraron a partir de los datos de 25 proyectos de IBM. El problema de esta medición es que sirve bien para medir el producto ya desarrollado pero no es útil para hacer estimaciones.

Para fines prácticos, se han elaborado diferentes tablas, usando datos históricos, de los cuales se desprende una correspondencia a muy grandes rasgos entre un punto de función 1 FP y 100 LOCs en un lenguaje de alto nivel. Pero hay que tomarlo con mucha reserva para hacer estimaciones de tamaño.

A continuación se mencionan observaciones comprobadas empíricamente por los autores mencionados entre corchetes, que explican la dificultad de hacer estimaciones para los proyectos de desarrollo de software:

1. **La productividad de los desarrolladores varía de manera considerable [Sackman]**
   Razón: El desarrollo de software es una actividad intelectual. Los seres humanos tenemos diferentes capacidades intelectuales y por lo tanto la productividad varía.
2. **Una gran cantidad de factores influye en la productividad de un desarrollador [Nelson-Jones].**
   Razón: El desarrollo de software es una actividad social. Los sistemas sociales tienen por lo general, una historia que conlleva múltiples factores y elementos que interaccionan de manera compleja.
3. **El esfuerzo de desarrollo es una función (no lineal) del tamaño del producto [Boehm].**
   Razón: El tamaño del producto influye en el esfuerzo pero también hay que tomar en cuenta la complejidad del código a desarrollar. Por ejemplo, los sistemas de control o compiladores pueden no ser de tamaño considerable, pero pueden requerir de un esfuerzo equiparable con un sistema de aplicación comercial de tamaño grande.
4. **La mayoría de las estimaciones de costo son demasiado bajas [DeMarco-Glass].**
   Razón: Las estimaciones se hacen al inicio del proyecto. Es muy natural que en este momento no se tomen en cuenta las actividades que al final no agreguen nada. Por ejemplo, las actividades relacionadas con modificaciones a causa de cambio de requerimientos, conocidas como re-trabajo.

   Seguimiento del Proyecto

   El propósito de dar el seguimiento al proyecto es asegurar que se cumplan sus objetivos. De manera periódica, se supervisa y evalúa el progreso para identificar las desviaciones con respecto a lo planeado y para acordar las acciones correctivas, cuando sea necesario. También, se identifican, analizan y controlan los riesgos.

   El problema con el seguimiento de proyectos es principalmente psicológico. A la gente no le gusta ser supervisada. Para minimizar tal problema se recomienda realizar el seguimiento a través de los mecanismos previamente establecidos, como pueden ser las reuniones semanales del equipo con el administrador o reportes de avance periódicos como se hacen con las forma semana personal.

   Una de las acciones correctivas, que se aplica frecuentemente en el caso de retraso del proyecto con respecto a las fechas planeadas, es incorporar más recursos humanos. A continuación presentamos una observación comprobada que explica la peligrosidad de este tipo de soluciones.

5. **Agregar recursos humanos a un proyecto atrasado lo atrasa más [Brooks].**
   Razón: La gente nueva no cuenta con el mismo conocimiento sobre el proyecto que tiene el equipo original. Para incorporar un miembro nuevo en el proyecto se

requiere de su capacitación, de una nueva división de tareas y se incrementa el costo de la comunicación dentro del equipo. Todo esto requiere de tiempo que atrasa más el proyecto.

Referencias bibliográficas

- Braude E. J. Ingeniería de software. Una perspectiva orientada a objetos. Alfaomega 2003.
- Cantor M.R., Object Oriented Project Management with UML. Wiley 1998.
- Endres A. y D. Rombach, A Handbook of Software and Systems Engineering, Empirical Observations, Laws and Theories, Pearson Education Limited, 2003.
- Humphrey Watts S., "Introduction to Personal Software Process", SEI Series in Software Engineering, Addison Wesley, 1997.
- Humphrey Watts S., "Introduction to Team Software Process", SEI Series in Software Engineering, Addison Wesley, 2000.
- Oktaba et al., Modelo de Procesos para la Industria de Software MoProSoft V 1.3, 2005, www.software.net.mx.

Capítulo 6

Fase de Especificación de requerimientos

Objetivos del capítulo:

El objetivo de este capítulo es plantear el proceso de la fase de especificación de requerimientos. Por lo que se pretende:

- Mostrar todas las actividades para hacer la especificación de los requerimientos funcionales y no funcionales de un producto de software.
- Aprender a definir las pruebas que demuestren que se cumplen los requerimientos del software especificados previamente.

6.1 Fase de Especificación de requerimientos: objetivos, actividades y productos.

El objetivo de esta fase es iniciar el desarrollo del producto de software. Por lo que se debe entender bien el problema a resolver para especificar los requerimientos del software y tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado, los cuales se incluyen en el Plan de pruebas del software.

El proceso de desarrollo a emplear, a partir de esta fase, está basado en tecnología orientada a objetos y los modelos se construyen con la notación de *Unified Modeling Language* (UML) [Booch 99].

Objetivo:

O1. Entender el problema a resolver.

Para construir un producto de software es necesario contar con la cooperación del cliente a fin de, entender cuál es el problema que desea resolver y cuáles son sus necesidades reales. El cliente usa términos de su entorno, cuyo significado no siempre es claro. Para evitar malos entendidos se establece un vocabulario común mediante un glosario de términos.

Actividades:

A1.1 Entender cuáles son las necesidades o requerimientos del software.

Producto:

- Definición del problema.
- Glosario de términos.

Objetivo:

O2. Proporcionar una descripción clara y no ambigua de las necesidades del software a través de la especificación de requerimientos funcionales y no funcionales.
Para lograr la especificación de requerimientos se emplean varias técnicas, como la identificación de casos de uso, prototipo de la interfaz de usuario y la identificación de los requerimientos no funcionales.

Actividades:

A 2.1 Especificar los requerimientos funcionales con casos de uso.
A2.2 Construir el prototipo la interfaz de usuario.
A2.3 Identificar los requerimientos no funcionales.

Producto:

- La Especificación de requerimientos del software, que incluye:
  - Diagrama general de casos de uso, indicando el alcance en el primer ciclo.
  - Detalle de los casos de uso del primer ciclo.
  - Prototipo de la interfaz de usuario.
  - Lista de requerimientos no funcionales.

Objetivo:

O3. Obtener criterios para validar el producto de software terminado.
Para saber si el producto que se desarrollará cumple con sus objetivos, hay que definir qué se espera de él, cómo se deberá comportar en situaciones normales y excepcionales.

Actividades:

A3.1 Construir el Plan de prueba del software.

Producto:

- Plan de prueba del software.

Formas a entregar en todas las fases:

- Semana del equipo
- Minuta de la reunión
- Lista de verificación
- Registro de defectos

Resumen de productos a entregar:

Producto	Responsable
Definición del problema	Administrador de desarrollo y Cliente
Glosario de términos	Administrador de desarrollo y Cliente
Especificación de requerimientos del software	Administrador de desarrollo
Plan de prueba del software	Administrador de desarrollo

6.2 Entender el problema

Para construir un producto de software, es necesario contar con la cooperación del cliente. Se debe entender cuál es el problema que tiene y cuáles son sus necesidades reales.

La mayor parte de las veces, el cliente no tiene claro qué es lo que realmente necesita. Es el desarrollador el responsable de ayudar al cliente a entender y expresar sus necesidades para que el software las pueda satisfacer. Para identificar los requerimientos se consultan a los interesados a través de varias técnicas [Tomayco] como son:

- Hacer entrevistas.
- Aplicar cuestionarios.
- Observar a los futuros usuarios al realizar las tareas que apoyará el software.
- Revisar documentos o sistemas ya existentes que se pretenden mejorar.

El cliente expresa de manera oral sus necesidades sobre el software que desea se le construya. Escribir un texto, entre el cliente y el desarrollador, que defina el problema, permite poner en blanco y negro las necesidades del software y ayuda al cliente a precisarlas. Por lo que la entrada inicial al proceso de desarrollo de software es escribir en un texto la *Definición del problema.*

El texto deberá escribirse preferentemente en un lenguaje que entienda el cliente sin términos técnicos computacionales.

Coad [Coad] propone una serie de estrategias al redactar ese texto, para ayudar a aclarar el objetivo del software. Una de ellas es escribir una frase del tipo "*para ayudar a"* o "*para apoyar a"* que permiten saber el tipo de usuarios que tendrá el software y para qué se usará. Otra estrategia que propone es que el desarrollador "*se dé una vuelta por el ambiente de trabajo donde se usará el software",* esto permite ver a los usuarios y el tipo de cosas que necesitan para que el sistema los apoye en su trabajo diario. Una estrategia más, indica que "*se haga una lista de las características o cualidades"* que deberá tener el software, la que se puede escribir en orden de importancia.

Por ejemplo, en la figura 6.1 se muestra una definición del problema para construir un sistema para una bolsa de trabajo en internet. En él se aplican dos de las estrategias propuestas por Coad, usar la frase "*para apoyar a"* y "*hacer una lista de características*".

DEFINICION DEL PROBLEMA:

Bolsa de trabajo

Se desarrollará un sistema de bolsa de trabajo *que apoye a* desempleados y a empresas. Este sistema podrá ser usado por desempleados que están en busca de algún empleo y por empresas que estén en busca de empleados para ocupar una vacante.

Los desempleados podrán ver las vacantes disponibles sin tener que registrarse, pero si el desempleado quiere postularse para una vacante tendrá que registrarse forzosamente. Las empresas podrán publicar sus vacantes por medio de un contacto, el cual será el responsable de los datos de la empresa, la empresa podrá revisar quién se registró para ocupar la vacante publicada. La empresa podrá contactarlo después de revisar su solicitud y el currículo del desempleado.

Lista de características deseadas

1. Cualquier usuario podrá ver las vacantes disponibles en el sistema de software.
2. El usuario que se interese por alguna vacante en el sistema y quiera postularse a ella tendrá que registrar sus datos y agregar su currículo.
3. Las empresas tendrán que tener una clave de usuario y una contraseña para poder ver los postulados para su vacante.
4. Las empresas tendrán un representante, el cual será el encargado de la administración de los datos de la empresa y sus vacantes.
5. Las empresas sólo podrán ver a los desempleados que se registraron para sus vacantes.
6. El sistema estará creado en ambiente Web y podrá ser accedido mediante cualquier explorador Web.
7. El sistema deberá ser fácil de usar aún sin tener ningún tipo de capacitación para utilizarlo.

Figura 6. 1. Ejemplo de Definición del problema.

La definición del problema permite establecer un acuerdo y entendimiento común entre el equipo de desarrollo y el cliente sobre lo que se va a hacer. En esta definición se sugiere que participen todos los miembros del equipo de desarrollo para entender el propósito del software que van a desarrollar.

Glosario de términos

La definición del problema es difícil, pues el cliente usa los términos relacionados con el problema y el técnico el lenguaje de los desarrolladores. Para que se puedan comunicar más fácilmente todos los involucrados, se recomienda construir un *Glosario de términos* que establece un vocabulario común.

El Glosario de términos es un pequeño diccionario, donde se pone cada término y su significado para este proyecto. En este glosario se deben incluir también los términos familiares para el desarrollador con su significado. La construcción del glosario es un trabajo continuo a lo largo del proyecto, se inicia en esta fase y se concluye al finalizar el proyecto. En la figura 6.2 se muestra un extracto del glosario de términos del sistema de bolsa de trabajo.

GLOSARIO DE TÉRMINOS

Sistema de Bolsa de trabajo

Actor: Es una entidad que interacciona con el sistema para obtener un resultado. Puede ser una persona, otro sistema, un dispositivo, etc.
Bolsa de Trabajo: En este caso, es la concentración de información acerca de empresas que tienen puestos disponibles (vacantes) y de personas que tienen la necesidad de conseguir un empleo.
Caso de uso: Es la descripción de un conjunto de secuencias de acciones que un sistema lleva a cabo para regresar un resultado observable a un actor.
Contraseña: Clave de seguridad que se le asigna a un usuario.
Desempleado: Persona que no tiene trabajo y está en busca de un empleo.
Empresa: Organización que tiene la necesidad de contratar a personas con capacidades especificas para su correcto funcionamiento.
Nombre de usuario: Identificación del usuario para poder acceder al sistema.
Vacante: Puesto disponible para ser ocupado por un desempleado por parte de una empresa.
Visitante: Persona que abre las páginas del sistema y revisa la información que se presenta en este sitio Web, la información la podrá ver sin tener que registrar sus datos.

Figura 6.2. Glosario de términos.

6.3 Especificación de Requerimientos.

Los requerimientos son el punto de partida para el desarrollo de un producto de software. *Los Requerimientos de software son un área de la Ingeniería de Software dedicada al estudio de la adquisición, análisis, especificación, validación y administración de los requerimientos o necesidades de un software.* [SWEBOK 2001]. Es una fase fundamental para el desarrollo de software con calidad. Se entiende por calidad en un producto de software aquel que cumple con las necesidades del usuario.

Un requerimiento o necesidad es lo que el cliente o un usuario necesitan que haga el software para resolver un cierto problema. Para definir los requerimientos deben colaborar conjuntamente varios roles: el Equipo de desarrollo, el Cliente (quien paga el software), los Usuarios (quienes lo usarán en su trabajo diario). Muchas veces el Cliente y el Usuario son la misma persona. A todas estas personas se les llama en inglés *stakeholders* que se traduce por *interesados* o *involucrados*.

La especificación de requerimientos es como un mapa para entender el problema a resolver. Los mapas son una ayuda para llegar más rápido al lugar deseado. Si no se tiene el mapa, se puede manejar a toda velocidad pero no necesariamente se llegará a la meta.

Una característica de los requerimientos es que cambian constantemente por muchas razones: se modifican las necesidades del cliente, cambia el ambiente, la tecnología, etc. por lo que, establecer los requerimientos es un proceso de negociación entre el cliente y los desarrolladores, donde ambos entienden y acuerdan el objetivo del software.

Los requerimientos deben formularse de forma clara, precisa y no ambigua. Para eso pueden usarse varias técnicas al mismo tiempo: el lenguaje natural, que es claro para el cliente pero ambiguo; el modelado gráfico, que es más claro para el desarrollador y no es ambiguo, pero puede no ser claro para el cliente; un prototipo de interfaz de usuario, útil para ambos pues es una representación visual de lo que hará el software.

Hay dos tipos de requerimientos: los funcionales y los no funcionales.

Los **funcionales** incluyen:

- Las entradas y salidas de datos, los cálculos o las funciones del software.

Los **no funcionales** son las características o restricciones que se esperan del sistema software:

- Necesidades de la interfaz externa como: tipo de usuario, hardware y software, comunicaciones, facilidades de uso requeridas por los usuarios.
- Atributos del software tales como: eficiencia, disponibilidad, seguridad, conversión, portabilidad, mantenimiento.
- Restricciones del diseño: de formatos, de archivos, lenguajes, estándares, compatibilidad.
- Otros: base de datos, instalación, etc.

El proceso para la **especificación de los requerimientos** es:

1. Entender las necesidades del software.
2. Identificar a los interesados en el sistema y solicitar los requerimientos.
3. Identificar y negociar los requerimientos funcionales y los no funcionales.
4. Hacer el prototipo de interfaz para que los interesados confirmen si esos son sus requerimientos.
5. Documentar la *Especificación de requerimientos*.

La especificación de requerimientos termina cuando se obtiene el documento de *Especificación de los requerimientos*, que resume lo que cliente necesita y que servirá de guía a los desarrolladores para analizar esos requerimientos, validarlos, administrarlos y generar el software. Validar un requerimiento implica comprobar que el software lo cumpla. Administrar los requerimientos significa que se lleve un control de los cambios que van surgiendo.

La Especificación de requerimientos es un documento donde se establece el acuerdo de lo que hará el software. Para hacer este documento se puede seguir lo que proponen los modelos de referencia, como MoProSoft [MoProSoft] para la estructura de este documento:

- Introducción. Es una descripción general del software y su propósito.
- Descripción de requerimientos.

6.3.1 Requerimientos funcionales.

Una de las técnicas para especificar los requerimientos funcionales es la de los *casos de uso,* los que servirán de hilo conductor de todo el proceso de desarrollo. Esta técnica se utiliza para identificar los requerimientos funcionales y a partir de ellos se diseña, implementa y prueba el software. Permiten rastrear los requerimientos a través de todo el proceso de desarrollo hasta el producto terminado.

Diagramas de caso de uso

Los casos de uso proporcionan una manera incremental y modular de describir software. Definen como los usuarios utilizarán el software. El conjunto de casos de uso conforma el modelo de casos de uso que describe el comportamiento general del sistema. Los casos de uso proporcionan una representación que puede ser fácilmente comprendida por todos los interesados. Se representan gráficamente con *Diagramas de casos de uso* de UML.

Elementos de los diagramas de casos de uso

Caso de uso. *Un caso de uso es una descripción de un conjunto de secuencias de acciones que realiza el sistema para entregar a un actor un resultado o valor observable*. [Booch, 1999]. Se representa por una elipse con el nombre del caso de uso. El nombre del caso de uso debe iniciarse, preferentemente, con un verbo en infinitivo y se recomienda expresar funcionalidad en términos familiares al cliente y a los futuros usuarios. En la figura 6.3 se muestra un ejemplo de caso de uso del sistema de bolsa de trabajo, en el que se expresa que el sistema debe contar con alguna facilidad para consultar una vacante.

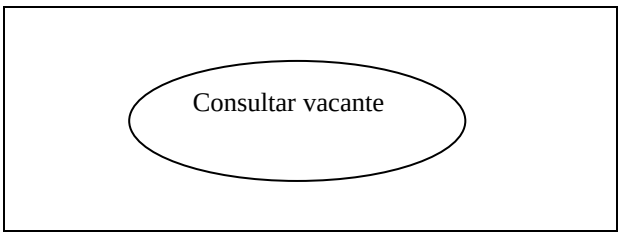


Figura 6.3. El caso de uso Consultar vacante para el sistema de bolsa de trabajo.

Actor. Es algo externo al software que intercambia información con el sistema, puede ser un usuario u otro sistema. El objetivo de un actor es completar una funcionalidad con el software para obtener un valor o servicio. Se representa con una figura humana con el nombre del actor en singular. Los sistemas externos con los que interacciona el software que se está desarrollando también son actores. Un caso de uso puede proporcionar un valor a uno o más actores. En el ejemplo del sistema de bolsa de trabajo (figura 6.4), un actor es la Empresa, quien representa a todas las posibles empresas que pueden administrar los datos de sus vacantes en el sistema o consultar los datos de los desempleados que se registraron para dichas vacantes.

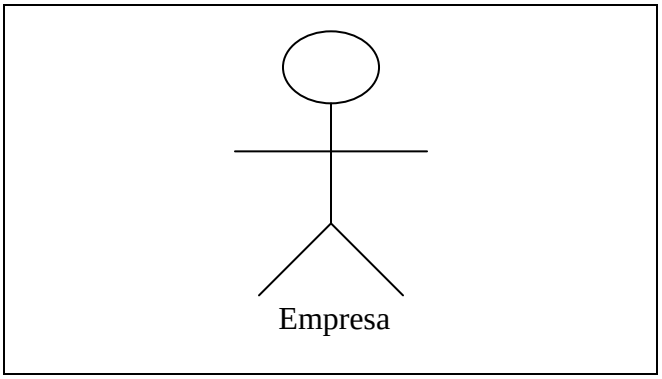


Figura 6.4. Un actor para el sistema de bolsa de trabajo.

Alcance del sistema. Representa la frontera del sistema y contiene los casos de uso que se realizan en cada ciclo de desarrollo. Se representa por un rectángulo que incluye los casos de uso que están dentro del alcance.

Diagrama de casos de uso. Un diagrama de casos de uso incluye los actores identificados, los casos de uso para cada actor y el alcance del sistema En el diagrama general de casos de uso se colocan las funcionalidades o casos de uso más generales, que posteriormente se detallan en diagramas en los que se desglosa cada funcionalidad. Este diagrama general tiene por objetivo mostrar de forma gráfica y clara las funcionalidades del sistema por lo que deberá ser simple, para mostrar a golpe de vista el alcance del sistema. Se recomienda que siempre se incluya un caso de uso para entrar al sistema y uno de salir para los actores. El diagrama general se discute con el cliente y los usuarios del software.

En la figura 6.5 se muestra el diagrama general de casos de uso del sistema de bolsa de trabajo. En este diagrama se aprecian 8 casos de uso generales que permiten entender la funcionalidad general del sistema y su alcance en el primer ciclo. Posteriormente cada uno de los casos de uso se detallará.

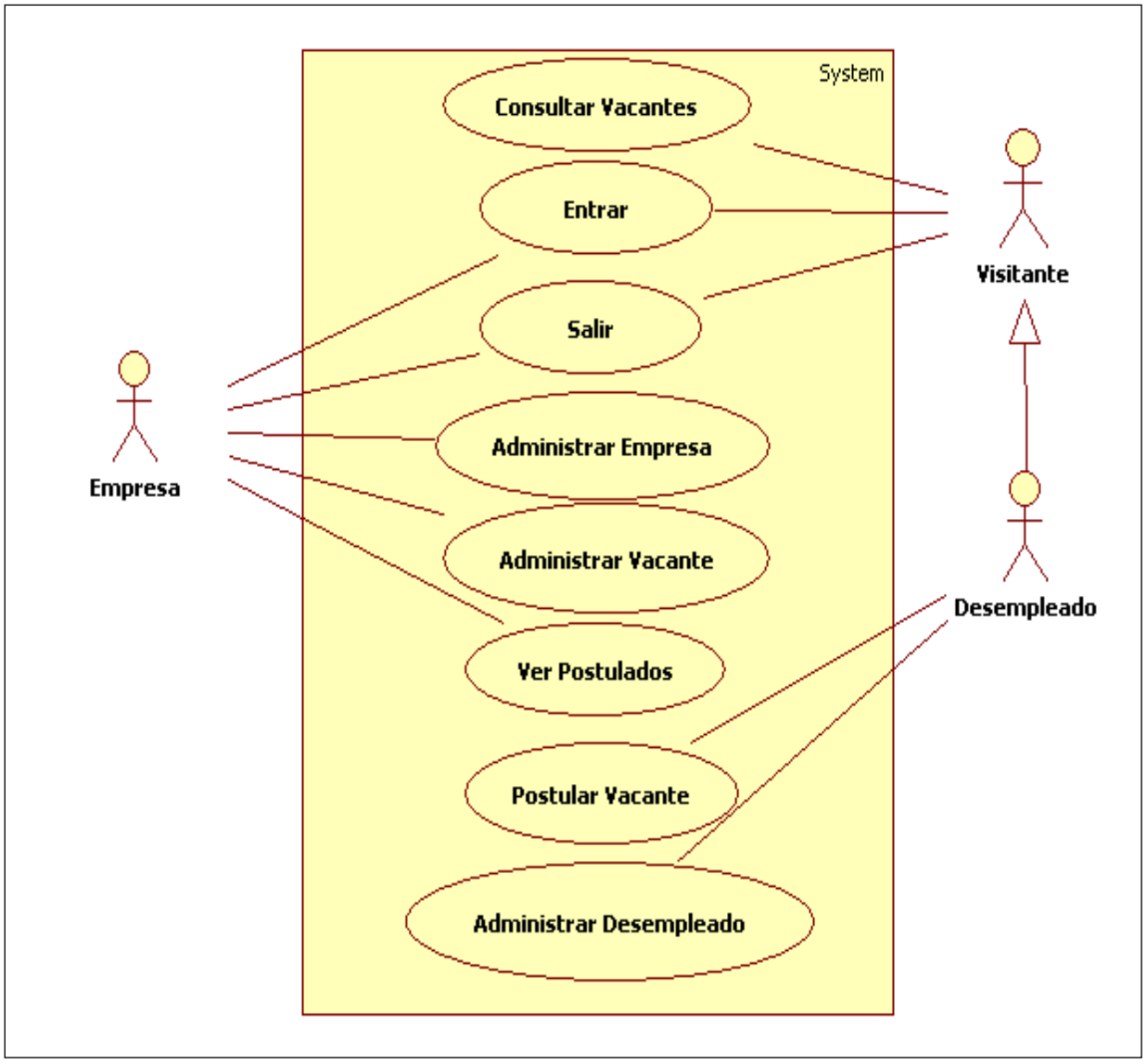


Figura 6.5. Diagrama general de casos de uso del sistema de bolsa de trabajo.

Proceso para la **creación de diagramas de casos de uso**:

1. Identificar los actores del sistema. Esto es, los que interaccionarán con el software para obtener un resultado de él.
2. Identificar las funcionalidades generales para cada actor y representarlos como casos de uso.
3. Definir el alcance o los casos de uso que se desarrollarán en el ciclo.
4. Detallar cada caso de uso.

Criterios para identificar a los actores:

1. Para identificar a un actor se revisa la definición del problema a fin de definir el tipo de usuarios u otros sistemas que interactuarán con el software. Se nombran con mayúscula y en singular.
2. Los candidatos para actores son roles generales no personas concretas. Por ejemplo Solicitante y no Juan Pérez.
3. Los casos de uso representan las necesidades de los usuarios del sistema que podrán ser modeladas y validadas en los casos de uso.
4. Para identificar los casos de uso, se revisa la definición del problema para cada actor, buscando las funcionalidades generales que requiere.

Detalle de los casos de uso

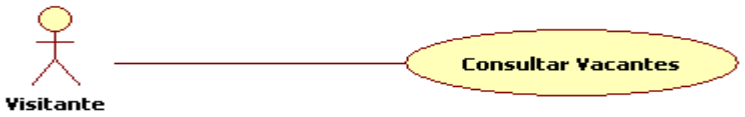
Una vez identificados los principales casos de uso para cada actor, se describe en detalle cada uno. La plantilla que se propone para esta descripción es:

> **Nombre**: El nombre deberá ser un verbo en infinitivo representativo de la funcionalidad del caso de uso.
> **Actor:** Nombre en singular del actor encargado de iniciar la acción y que recibe el resultado del caso de uso.
> **Diagrama detallado** del caso de uso indicando el actor y sus variantes o detalles.
> **Descripción:** Texto breve describiendo la acción.
> **Precondiciones**: Acciones previas del estado del sistema para que se pueda iniciar el caso de uso.
> **Flujo de eventos normales**: Tabla que describe el flujo de acciones entre el actor y el sistema durante el caso de uso.
> **Flujo de eventos alternativos o excepcionales**: Tabla con las acciones que ocurren en situaciones anormales o excepcionales.
> **Poscondiciones**: Define el estado del sistema después de la terminación exitosa del caso de uso.

En la figura 6.6 se muestra el detalle del caso de uso Consultar vacante del sistema de bolsa de trabajo.

CASO DE USO: CONSULTAR VACANTE

Actor: **Visitante**



Descripción: Un **Visitante** entra al sistema para consultar las vacantes que están disponibles.

Precondiciones:

- El visitante conoce la dirección Web del sistema de software SIBOT (nombre del sistema de bolsa de trabajo).
- El visitante desea ver las opciones de trabajo que están disponibles.

Flujo de eventos normales:

Usuario		Sistema		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	El usuario selecciona el vínculo Ver Vacantes.	2	Muestra la pantalla de Todas las Vacantes.	E1
3	Selecciona la Vacante de la cual desea ver los datos a más detalle.			
4	Oprime el botón Ver	5	Se muestra la pantalla de Detalle de Vacante con los datos de la vacante seleccionada.	E1,E2

Flujo de eventos alternativos:

Id	Nombre	Acción
E1	La conexión con la base de datos no está establecida o se interrumpió.	Se manda un mensaje de error, el cual indica que los datos no se pueden mostrar debido a que no hay conexión con la base de datos.
E2	No se ha seleccionado ninguna vacante	No hace nada.

Poscondiciones:

- Los datos de la vacante se muestran a detalle.

Figura 6.6. Detalle del caso de uso Consultar Vacante del sistema de bolsa de trabajo.

6.3.2 Prototipo de interfaz de usuario

Un prototipo de interfaz de usuario es una representación parcial de la interfaz de usuario que tendrá el software, que muestra la forma en que el usuario interaccionará con él. Este prototipo es un elemento muy importante para la comunicación con el usuario en la definición de los requerimientos, pues al revisar la interfaz, el usuario puede refinar sus necesidades y comunicarlas al desarrollador.

Se llama *pantalla* o interfaz al mecanismo con el cual interactúa el usuario con el sistema. Cuando se diseñen estas *pantallas* podrán ser código html, o ventanas o entradas de modo texto.

Para diseñar el prototipo de la interfaz se consideran los casos de uso planteados en el diagrama general y se puede construir al mismo tiempo que se detallan los casos de uso. Si el diagrama general tiene los casos de uso X, Y y Z, en la pantalla principal del sistema deberán estar las opciones del menú X, Y y Z. Si en la descripción del flujo de un caso de uso dice “el sistema presenta la pantalla de X...”, en el prototipo deberá existir esa pantalla. Si en el flujo dice “el usuario oprime el botón M...”, deberá haber un botón M en la pantalla.

En resumen, se debe cuidar que exista coincidencia entre el detalle de los casos de uso y el prototipo. Deben coincidir:

- Las opciones del menú y los casos de uso
- Los nombres de las pantallas
- Los nombres de los botones

En la figura 6.7 se muestra un ejemplo del prototipo de la interfaz del detalle de caso de uso **Alta de Vacante**.

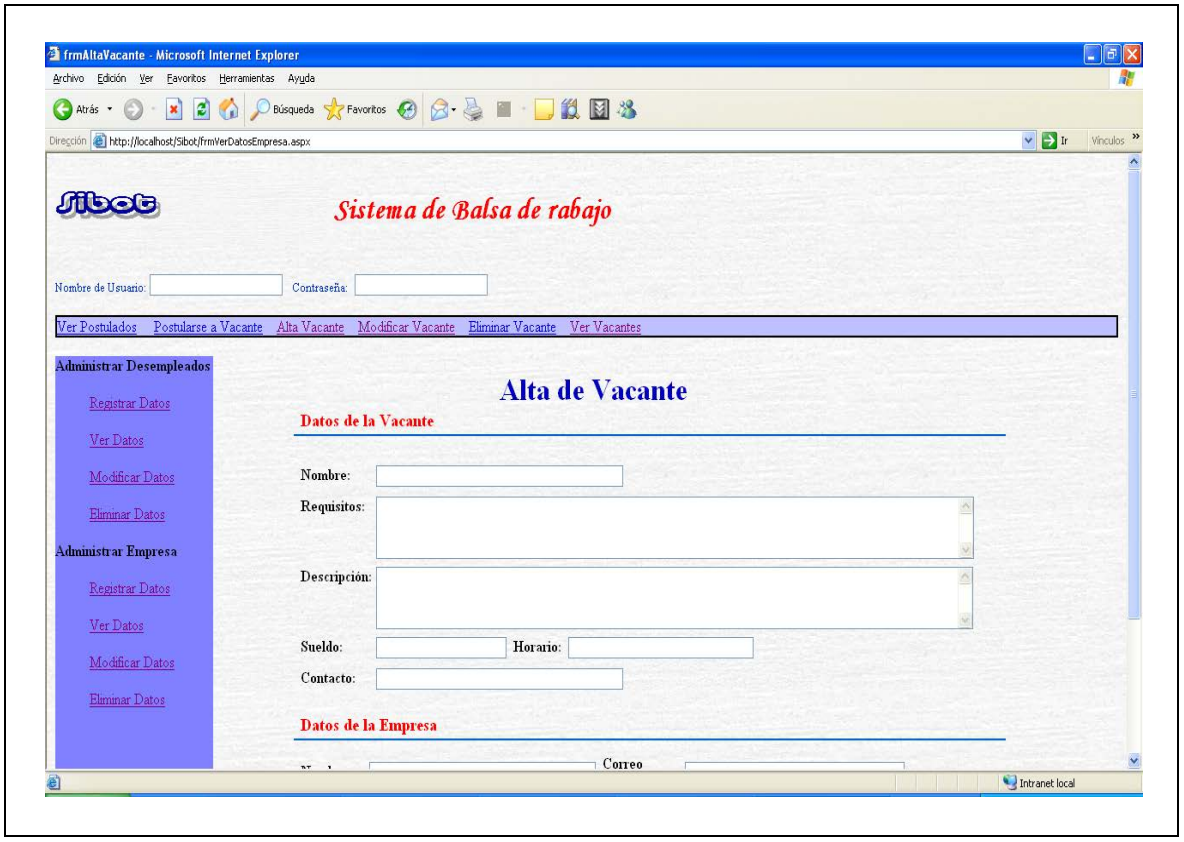


Figura 6.7. Pantalla para el caso de uso Alta de Vacante.

6.3.3 Requerimientos no funcionales

Los requerimientos no funcionales tienen que ver con los atributos o características que el cliente solicita para el funcionamiento del software.

Los requerimientos no funcionales pueden referirse a:

- **Interfaz con el usuario:** descripción de las características que permitan al software apoyar al usuario en sus tareas.
- **Interfaz externa**: relación o conexión con otros sistemas.
- **Confiabilidad**: solicitud del desempeño respecto a seguridad, tolerancia a fallas y su recuperación.
- **Eficiencia:** límites de tiempo de respuesta.
- **Mantenimiento**: facilidad de comprensión y realización de modificaciones futuras.
- **Portabilidad**: facilidad de transferencia de un ambiente de ejecución a otro.
- **Restricciones de diseño y construcción:** las que imponga el cliente.
- **Legales y reglamentarios:** necesidades impuestas por leyes o reglamentos de otros.

Los requerimientos no funcionales especificados para el sistema de la bolsa de trabajo se muestran en la figura 6.8.

REQUERIMIENTOS NO FUNCIONALES

Interfaz con usuarios

- El sistema debe de ser fácil de usar, además deberá tener una interfaz gráfica agradable y con colores suaves para no dañar en algún sentido visual al usuario.
- El funcionamiento del sistema deberá estar activo las 24 horas del día y los 365 días del año.

Confiabilidad

- Este sistema deberá de tener una alta confiabilidad e integridad con los datos de los usuarios.
- Para poder modificar, dar de alta o eliminar datos, el usuario debe de proporcionar al sistema un nombre de usuario y una contraseña.

Eficiencia

- La velocidad para mostrarle los datos al usuario debe de ser considerable, lo cual implica que la pagina no debe de contener imágenes que haga que el sistema se retarde.

Restricciones de diseño y construcción

- El sistema de software será construido para funcionar en ambiente Web.
- El sistema deberá estar codificado en lenguaje de programación Java.
- La base de datos del sistema de software deberá estar construida con el manejador de bases de datos MySQL.
- Para el desarrollo de este sistema se utilizarán las siguientes herramientas
    - ArgoUML para modelado del sistema.
    - Java para la codificación.
    - Microsoft Word para hacer la documentación.
    - Tomcat para el desarrollo del sitio web.
    - MySQL para la creación de la Base de Datos.

Figura 6.8. Requerimientos no funcionales para el sistema de bolsa de trabajo.

6.4 Plan de prueba del software

El objetivo del *Plan de prueba del software* es identificar las pruebas que son necesarias para asegurar el cumplimiento de los requerimientos especificados. Uno de los objetivos de especificar los requerimientos es poder rastrearlos a lo largo del desarrollo y validar que el producto que se construya cumple con ellos. Se mencionó que los casos de uso facilitan este rastreo, por lo tanto en el Plan de prueba del software se especifican casos de prueba para todos los casos de uso en situación normal y excepcional. De esta manera se tendrá la seguridad que el software cumple con las necesidades del cliente y no falta ni sobra ninguna funcionalidad.

La definición del Plan de prueba del software se puede realizar a la par del detalle de los casos de uso. Este plan consiste en una lista de casos de prueba para el flujo normal y para los flujos excepcionales de cada caso de uso. Al diseñar los casos de prueba se pueden hacer ajustes al detalle de los casos de uso pues se entienden mejor los requerimientos.

Los casos de prueba para cada caso de uso se documentan en una tabla. Un caso de prueba identifica las entradas al caso de uso y los resultados esperados. Se consideran entradas para flujos de eventos normales y eventos alternativos o excepcionales. En la figura 6.9 se muestran los casos de prueba que se determinaron de acuerdo a los flujos del caso de uso Consultar vacante del sistema de bolsa de trabajo.

CASO DE USO: CONSULTAR VACANTE

Caso de Prueba 1.	
Entradas	**Resultado esperado**
Selección de la opción Ver Vacantes, por parte del usuario. Selección del botón Ver de la vacante seleccionada por parte del usuario.	Pantalla Todas las Vacantes. Detalle de la vacante seleccionada
Caso de Prueba 2.	
Entradas	**Resultado esperado**
Selección de la opción Ver Vacantes, por parte del usuario.	Mensaje de error que indica que los datos no se pueden mostrar debido a que no hay conexión con la base de datos.

Figura 6.9. Casos de prueba para el caso de uso Consultar vacante del sistema de bolsa de trabajo.

6.5 Profundización de los temas para el segundo ciclo

6.5.1 Captura de requerimientos ágiles

Los métodos ágiles son los procesos de desarrollo de software que se consideran menos *burocráticos* en cuanto a la cantidad de documentación que se debe generar y mas orientados a las personas. Otra característica importante de estos métodos es que son iterativos e incrementales de igual manera como son TSPi [Humphrey] y el Proceso Unificado [Jacobson]. La diferencia fundamental entre estos dos métodos y los llamados ágiles, es que consideran iteraciones más cortas, de 3 o 4 semanas en las que se entregan productos parcialmente ejecutables al cliente. Ejemplos de estos métodos son XP, SCRUM y otros [Larman].

Los procesos de desarrollo de métodos ágiles tienen como algunos de sus principios [Manifiesto Ágil] los siguientes:

- La satisfacción del cliente
- Considerar el constante cambio de los requerimientos
- Entregar un producto funcional con frecuencia

- El trabajo conjunto con los clientes
- La forma mejor de obtener los requerimientos es con reuniones frecuentes cara a cara con los clientes.

Es una interpretación errónea de estos principios que no deban escribirse ni modelarse los requerimientos. En las primeras iteraciones, el principal producto que se obtiene es la clarificación de los requerimientos (hasta 80% en las primeras 4 iteraciones) y un 10% en la construcción del producto. [Larman]

Para la especificación de los requerimientos se propone no generar documentos tan formales como los aquí mencionados. Se proponen varias técnicas diferentes a las establecidas en este capítulo, pero todas llevan a definir con claridad las necesidades del software.

Algunos métodos usan los casos de uso detallados en forma de escenarios que se implementan contemplando los aspectos de la arquitectura, las interfaces de usuario hasta el manejo de la base de datos. [Cockburn]

Las *historias de usuario* las propone XP como método para capturar y priorizar los requerimientos. En una tarjeta se escribe la necesidad de usuario y se asigna a un programador quien lo comenta con el cliente hasta que está listo para generar el código necesario.

Otra técnica es hacer el prototipo de la interfaz de usuario en papel y representar las pantallas o páginas web en un pizarrón en el que se pega una representación de la interfaz e indicando la navegación entre ellas. Esta técnica se llama *GUIs con pegamento* que permite capturar los requerimientos y la visión de cómo se interactúa en el sistema.

Para otras técnicas se puede consultar [Larman].

Como se puede apreciar, las técnicas y modelados expuestos en este capítulo son útiles en cualquier enfoque de procesos de desarrollo que se adopte ya sea ágil o disciplinado.

6.5.2 Estándar de especificación de requerimientos de MoProSoft

Según MoProSoft la especificación de requerimientos se compone de una introducción y una descripción de requerimientos:

Introducción: Descripción general del software y su uso en el ámbito de la organización
Descripción de requerimientos:

- Funcionales: necesidades que debe satisfacer el software.
- Interfaz de usuario: definición de las características de la interfaz de usuario incluyendo la descripción del prototipo.
- Interfaces con otro software o hardware

- Confiabilidad: nivel de desempeño con respecto a la tolerancia a fallas y recuperación.
- Eficiencia: desempeño del software con respecto al tiempo y utilización de recursos.
- Mantenimiento: elementos que facilitarán las modificaciones del software.
- Portabilidad: características que permitan su transferencia a otros ambientes.
- Restricciones del diseño y construcción: necesidades impuestas por el cliente.
- Legales y reglamentarios: necesidades impuestas por leyes y reglamentos.

Referencias bibliográficas

- Booch G., J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley. 1999
- Coad P. at. al. *Object Models, Strategies, Patterns and Applications*. Yourdon Press Computing Series, Prentice Hall. 1995
- Cockburn A. *Agil Software Development*. Addison Wesley 2002.
- Endres A., Rombach D. *A Handbook of Software and Systems Engineering.Empirical Observations, Laws and Theories.* Pearson/Addison Wesley 2003.
- Humphrey, W. *Introduction to Team Software Process*. SEI Series in Software Engineering Addison Wesley. 2000
- IEEE estándar 830 de 1984.
- Jacobson I., Booch G., J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley. 1999
- Larman C. *Agile and Iterative Development. A manager's Guide*. Addison Wesley 2004.
- MoProSoft Modelo de Procesos para la Industria de Software. Versión 1.3 Agosto 2005. www.software.com.mx
- Pressman Roger S., "Ingeniería del software. Un enfoque práctico". McGraw Hill 3ª Ed. 1993.
- SWEBOK. *Guide to the Software Engineering Body of Knowlwdge*. Trial version Mayo 2001. www.swebok.org
- Tomayko J. E., Hazzan O. Human Aspects of Software Engineering. Charles River Media, Computer Engineering Series. 2004.

Capítulo 7

Fase de Diseño

Objetivos del capítulo:

Describir las actividades necesarias para diseñar la arquitectura del sistema, identificar los elementos con los que se construirá, como se estructurarán y preparar el Plan de pruebas de integración.

7.1 Fase de Diseño: objetivos, actividades y productos

Los objetivos de esta fase son: conocer en qué consiste el diseño del software para el trabajo en equipo, definir la arquitectura al identificar y describir las partes que compondrán el software con sus relaciones, así como, establecer las clases con que se construirá cada parte, utilizando la vista estática (diagramas de clases) y la dinámica (diagramas de secuencia y de estados). Además de elaborar el Plan de pruebas de integración.

Objetivo:

O1. Conocer el proceso del diseño para el trabajo en equipo.
Los productos que se elaboran en la fase de diseño servirán como base para la construcción de la solución computacional al problema planteado, es decir el sistema. El diseño tiene unos principios importantes a conocer sobre todo cuando se trabaja en equipo. En esta fase, también, se decide el ambiente de implementación y se define el estándar de documentación del diseño.

Actividades:

A1.1 Entender los principios del diseño para trabajo en equipo.
A1.2 Decidir las tecnologías y el ambiente de implementación.
A1.3 Establecer el estándar de diseño.

Productos:

- Especificación del ambiente de implementación.
- Estándar de documentación del diseño

Objetivo:

O2. Definir la arquitectura del software.
La arquitectura establece las partes principales del software a partir de los casos de uso de la especificación de requerimientos. Si el software se ejecutará en más de una computadora se debe construir el diagrama de distribución indicando qué partes o componentes le corresponde a cada una.

Actividades:

A2.1 Identificar la arquitectura del software.
A2.2 Hacer el diagrama de distribución.

Productos:

- Arquitectura con diagrama de paquetes.
- Diagrama de distribución.

Objetivo:

O3. Diseñar los componentes principales del software a través de clases.
Para cada componente se identifican las clases necesarias para su construcción, se modelan la vista estática (diagrama de clases) y la dinámica (diagramas de secuencia y de estados).

Actividades:

A3.1 Identificar las clases importantes a relacionarlas a través de diagramas de clases, constituyendo la vista estática.
A3.2 Modelar la participación dinámica de dichas clases en la realización de los casos de uso con diagramas de secuencia.
A3.3 Modelar por medio de diagramas de estado la navegación en la interfaz de usuario o la secuencia válida de casos de uso en el sistema.

Productos:

- Diagramas de clases.
- Diagramas de secuencia.
- Diagrama de estados.

Objetivo:

O4. Definir el orden de la integración de los componentes y su prueba.
Una vez definida la arquitectura y detallados los componentes, se establece el orden en que se integrarán para formar el producto de software.

Actividades:

A4.1 Crear el Plan de pruebas de integración.

Productos:

- Plan de prueba de integración.

Formas a entregar en todas las fases:

- Semana personal
- Semana del equipo
- Minuta de la reunión
- Lista de verificación
- Registro de defectos

Resumen de productos a entregar:

Producto	Responsable
Especificación del ambiente de implementación	Administrador de desarrollo y el de Apoyo
Estándar de documentación del diseño	Administrador de desarrollo y el de Apoyo
Arquitectura con diagrama de paquetes	Administrador de desarrollo
Diagrama de distribución	Administrador de desarrollo
Diagrama de clases	Administrador de desarrollo
Diagramas de secuencia	Administrador de desarrollo
Diagrama de estados	Administrador de desarrollo
Plan de prueba de integración	Administrador de desarrollo

7.2 Diseño

"El diseño de software es un proceso creativo para decidir cómo se construirá el producto de software" [Humphrey]. Los objetivos del diseño son: identificar y caracterizar las partes o componentes principales del software, definir su interacción e integración en el producto. En esta fase el enfoque es definir cómo se construirá el sistema.

El proceso del diseño tiene dos niveles de abstracción: el arquitectónico y el detallado. El diseño arquitectónico identifica los componentes principales partiendo de la especificación de requerimientos. El diseño detallado especifica en detalle los componentes tomando en cuenta el ambiente en que se codificará.

7.2.1 Principios del diseño

Algunos principios del diseño [Humphrey] son:

Diseñar para el cambio. Como se indicó en los principios de la Ingeniería del software el software cambia constantemente, por lo que es fundamental anticiparse a los cambios. Esto significa que el diseño debe ser flexible para permitir cambios con relativa facilidad.

Diseñar para facilitar el uso del software. Es importante diseñar teniendo en mente a los usuarios del software y sus aptitudes. Considerar algunos escenarios del uso del software, puede ayudar en la identificación de los componentes que deberá tener.

Diseñar para facilitar la prueba. Este principio está enfocado en el desarrollador que probará el sistema. Se identifican los componentes del sistema como unidades que se puedan probar sin necesidad de incluir a otros componentes.

Diseñar para la reutilización. Una manera de mejorar la productividad del equipo en proyectos futuros o en ciclos siguientes, es definir partes genéricas que puedan volver a usarse. Para aplicar este principio, se deben identificar los componentes comunes que se podrán reutilizar. El reuso incluye no solo el nivel del diseño, sino de código, casos de prueba, modelos o diagramas.

Para apoyar la aplicación estos principios en el diseño, se tienen dos medidas que ayudan a estructurar e identificar los componentes:

- **Cohesión**. Es el grado de relación entre los elementos que pertenecen a un componente. Un buen diseño tiene un grado de cohesión alto entre los elementos de sus componentes. El libro de [Pressman] proporciona algunos criterios sencillos para medir el grado de cohesión de un componente: "Escribir una frase que describa el objetivo de un componente. Si la frase tiene un solo verbo, el componente tiene un buen grado de cohesión. Si la frase es compuesta, contiene más de un verbo o contiene comas, probablemente tiene una cohesión débil y habría que definir un componente para cada verbo".
- **Acoplamiento**. Es el grado de relación entre los componentes. Un buen diseño tiene un acoplamiento bajo entre sus componentes. Esto es, cada componente se relaciona con otros con pocas interacciones.

Un diseño es bueno si la cohesión de sus componentes es alta y el acoplamiento entre ellos es débil.

Los principios del diseño pueden usarse como criterios para evaluar los diseños. Es importante aclarar que no existe *El* buen diseño.

Especificación del ambiente de implementación

Dependiendo del tipo de aplicación a desarrollar, se selecciona el ambiente y herramientas con que se hará la programación. Se debe definir:

- El ambiente de desarrollo y su versión.
- Las herramientas de diagramación para UML.
- Si se requiere un manejador de bases de datos, cuál se usará y en qué versión.
- Si la aplicación es para web, se define el servidor de aplicaciones.
- Herramientas para automatizar las pruebas unitarias.
- Etc.

Una vez definido el ambiente de implementación, el administrador de apoyo consigue el software y lo instala en las máquinas del equipo de trabajo.

Estándar de documentación del diseño

Se define el estándar de documentación del diseño para facilitar a los miembros del equipo trabajar por separado e integrar sus documentos sin dificultad. Este estándar incluye:

- Convención para el nombrado de componentes, clases, variables, métodos, etc. Se definen los nombres de los componentes de forma que muestren la jerarquía a la que pertenecen: el sistema, componente u objeto. Las convenciones usadas para nombrar los programas, archivos, variables, parámetros. Esto ayuda al equipo a saber cómo se llamarán los componentes que desarrollarán sus compañeros. Estas convenciones se pueden basar en el estándar de implementación de SUN para programas en Java.
- Estándar de diseño de la interfaz con el usuario: página principal con opciones de menú, diseño gráfico, colores, nombres de botones, títulos, tipo de letras etc.
- Estándar de mensajes de error, que mandará el sistema.

7.3 Arquitectura de software

La arquitectura de software es el diseño del más alto nivel. Consiste en definir cuáles serán los componentes que formarán el software. La arquitectura debe favorecer el cumplimiento de los requerimientos funcionales y no funcionales especificados para el producto.

Las cualidades que debe tener la arquitectura son:

- Sencillez. Que sea fácil de comprender y de implementar.
- Extensión. La posibilidad de agregar nuevos componentes.
- Cambio. Que los cambios en los requerimientos no afecten mucho a la arquitectura.

El modelo de capas, es la base de la arquitectura de un sistema cuando es posible estructurar la solución en grupos de tareas, donde cada grupo tiene un nivel de abstracción particular.

Una *capa* es una abstracción que toma el resultado de la capa inferior, efectúa su función y entrega ese resultado a la capa superior. La regla en este modelo es que cada capa se comunica solamente con las capas adyacentes.

Para definir la arquitectura se usarán las siguientes capas:

Capa de Presentación. En esta capa se colocan los elementos con los que interactúa directamente el usuario: las ventanas, páginas, informes, etc.
Capa Lógica de la aplicación. Son los elementos que implementan las reglas del negocio y la lógica de la aplicación.

Capa de Almacenamiento. Contiene los elementos que guardan la información para asegurar su persistencia tales como bases de datos o archivos.

Una vez definida la arquitectura del producto de software, se representa con diagramas de paquetes de UML.

Diagrama de paquetes

Un paquete es un mecanismo general de UML para organizar elementos en grupos. Los paquetes sirven para representar las capas de la arquitectura.

Un paquete se representa con una carpeta con su nombre. Los paquetes pueden contener otros paquetes, clases, interfaces, código html, etc. Los elementos de cada paquete deben tener una alta cohesión y bajo acoplamiento con los elementos de otros paquetes.

Un diagrama de paquetes se construye mostrando las relaciones entre ellos.

Tipos de relaciones entre paquetes:

Dependencia (se representa por una línea punteada). Un paquete B depende de otro paquete A, si se presenta un cambio en A, puede afectar al que depende de él, es decir B. En este caso, la flecha de dependencia parte de B y apunta al paquete A.

Asociación (se representa por una línea continua). Establece una relación entre dos paquetes que requieren de los servicios uno del otro. Esta relación es bidireccional.

Generalización (se representa por una línea continua con triángulo transparente que apunta hacia el paquete más general). Un paquete representa los aspectos más generales y otro los especializa, es la relación de herencia.

Realización (se representa por una línea punteada con triangulo transparente que apunta hacia el paquete que va a ser realizado por otro). Es un contrato que promete que un paquete va a ser implementado por otro.

Una arquitectura por capas constituye un punto de partida para distribuir los componentes que formarán al software. Las agrupaciones de estos se pueden representar por paquetes. Por lo que, se puede distribuir paquetes a lo largo de las 3 capas antes mencionadas.

- **Capa de Presentación.** Esta capa puede contener uno o varios paquetes que contengan la interfaz de usuario. Por ejemplo, el paquete de la interfaz en el servidor y el paquete con la interfaz del cliente. Los paquetes de la capa de Presentación tienen el sufijo **IH.**
- **Capa de Lógica de la aplicación.** En general, esta capa contiene varios paquetes, uno para cada funcionalidad del sistema. Estos paquetes pueden tener relaciones de dependencia, asociación o generalización.

- **Capa de Almacenamiento**. En este paquete puede estar la base de datos o archivos en uno o más paquetes.

Las relaciones entre estos tres paquetes serán de asociación del paquete de nivel inferior (Almacenamiento), al intermedio (Lógica de la aplicación) y de éste al superior (Presentación).

En la figura 7.1 se muestra un diagrama con las 3 capas de la arquitectura. Los paquetes de la capa de Presentación son *DesempleadosIH, VacantesIH* y *EmpresaIH*. Los siguientes paquetes pertenecen a la capa de Lógica de la aplicación, las relaciones que se muestran entre estos paquetes son dependencias. Por ejemplo el Paquete *Postulaciones* depende de *Vacantes* y de *Desempleado.* El paquete de *Base de Datos* se encuentra ubicado en la capa del Almacenamiento.

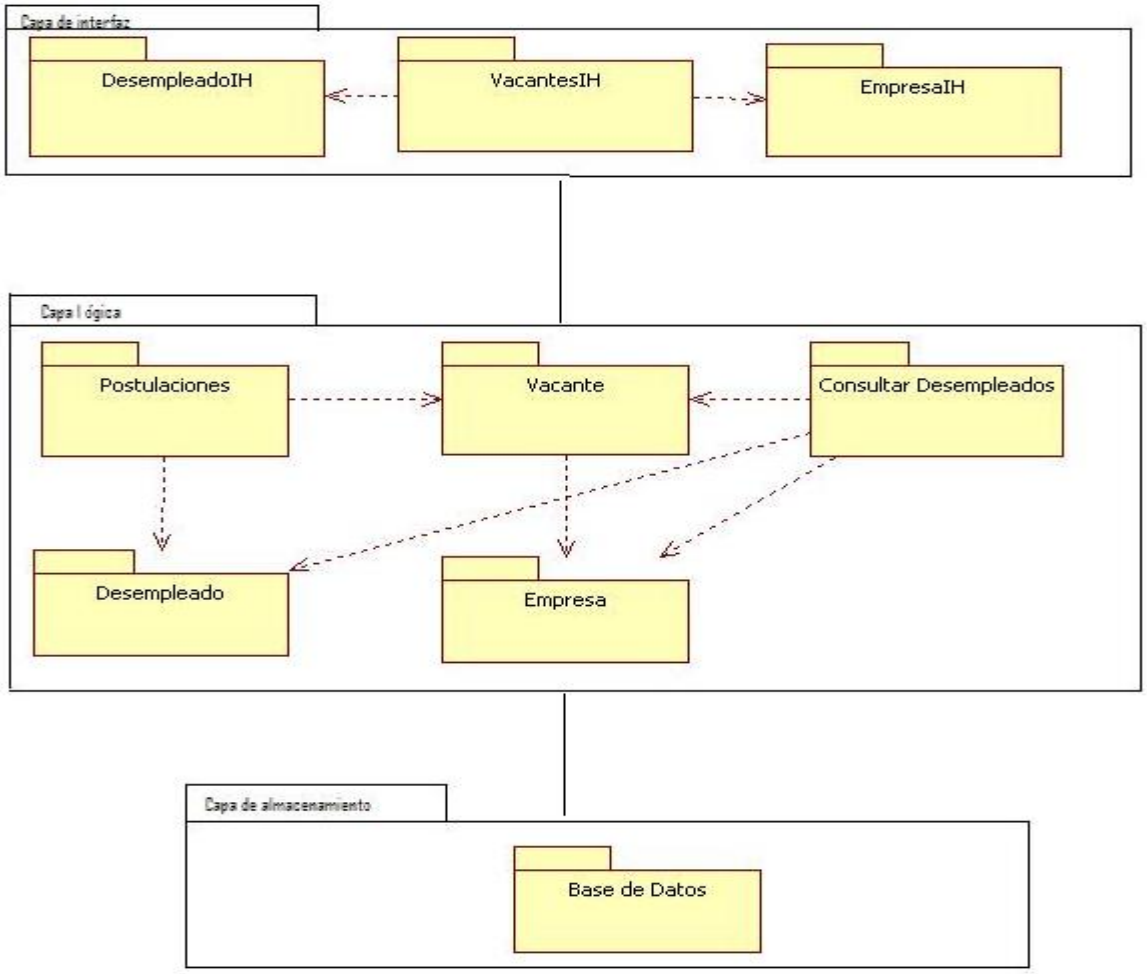


Figura 7. 1. Diagrama de paquetes para el sistema de bolsa de trabajo.

Diagrama de distribución

Un sistema es distribuido si se ejecuta en más de una computadora y sus componentes se encuentran distribuidos en ellas. Los diagramas de distribución representan los componentes que se instalarán en cada máquina y las conexiones entre ellos. Para mostrar este aspecto de la arquitectura, UML tiene los diagramas de distribución (*deployment*).

Los elementos de estos diagramas son: los *nodos* y las *conexiones* entre ellos.

Un *nodo* es un elemento físico que existe y representa un recurso computacional [Booch]. Se representa por un cubo que debe nombrarse.

Las *conexiones* son relaciones que representan las comunicaciones entre los nodos. Pueden etiquetarse con un protocolo de comunicación.

En la figura 7.2 se muestran los nodos de un sistema web. La conexión entre servidor y el cliente es por el protocolo http.

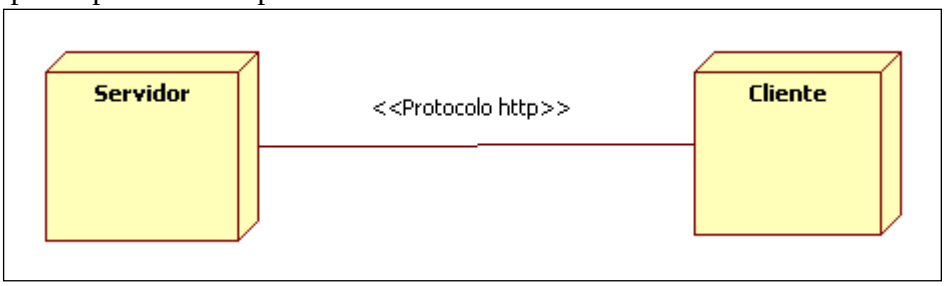


Figura 7. 2. Ejemplo de diagrama de distribución para un sistema web.

Proceso para definir la arquitectura:

1. Seleccionar el tipo de arquitectura del software según la aplicación. Inicialmente se puede elegir la arquitectura de capas. Posteriormente, se puede revisar otras arquitecturas alternativas.
2. Identificar los paquetes del sistema.
3. Construir el diagrama de paquetes.
4. En el caso de que sea un sistema distribuido, identificar la lógica de la distribución.
5. Construir el diagrama de distribución y asignar los paquetes a los nodos.

7.4 Diseño de clases

Para avanzar en el nivel de detalle del diseño se requiere la identificación de los elementos de cada paquete de la arquitectura. En el caso del diseño orientado a objetos los elementos de los paquetes serán clases. Se construyen dos vistas del diseño: la *vista estática* formada por *diagramas de clase* que representan los elementos estructurales y sus relaciones. La *vista dinámica* con las interacciones de los objetos de esas clases que se modela con *diagramas de secuencia***.**

7.4.1 Vista estática.

Para identificar distintos tipos de clases se usan tres estereotipos:

Clases de interfaz (interfaz con el usuario). Son los elementos con los que interactúa directamente el usuario: las ventanas, páginas, informes, etc. Estas clases corresponden a la capa de Presentación. Al identificar estas clases, se les puede poner al nombrarlas el sufijo **IH** para diferenciarlas de las clases de los otros estereotipos.
Clases de control (reglas del negocio o aplicación). Son los elementos que implementan las reglas del negocio y la lógica de la aplicación y corresponden a la capa de Lógica de la aplicación.
Clases de entidad (bases de datos, archivos). Son los elementos que guardan la información para asegurar su persistencia. Pertenecen a la capa de Almacenamiento. Estas clases pueden tener en su nombre el sufijo **BD**.

Estos tres estereotipos están definidos por UML para facilitar la identificación de las clases del sistema que se incluyen en los diagramas de clases, que forman la vista estática de los componentes que formarán el software.

Diagramas de clases

Los diagramas de clases se usan para modelar gráficamente la vista estática del software. Describen los tipos de objetos que son importantes para modelar un sistema y cómo se relacionan [Arlow]. Contienen los siguientes elementos:

- **Clases** es la descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y comportamiento [Rumbaugh]. Se representa gráficamente por un rectángulo con usualmente 3 compartimentos, uno para el nombre, otro para los atributos y el tercero para los métodos. En la figura 7.4 se muestra la representación gráfica de la clase Desempleado del sistema de bolsa de trabajo.

Figura 7. 3. Representación de clase Desempleado en UML, del sistema de bolsa de trabajo.

- **Relaciones** muestra la dependencia entre dos o más clases. Los tipos principales de relaciones entre clases son: asociación, agregación y generalización.

  - **Asociación** es una relación estructural que describe ligas entre objetos de las clases. Se representa por una línea que conecta a las clases asociadas. Esta liga puede tener adornos como la multiplicidad que denota cuántos objetos de una clase pueden estar relacionados con objetos de la otra. En la figura 7.5 se muestra un ejemplo de asociación entre las clases Desempleado y Vacante del sistema de bolsa de trabajo.

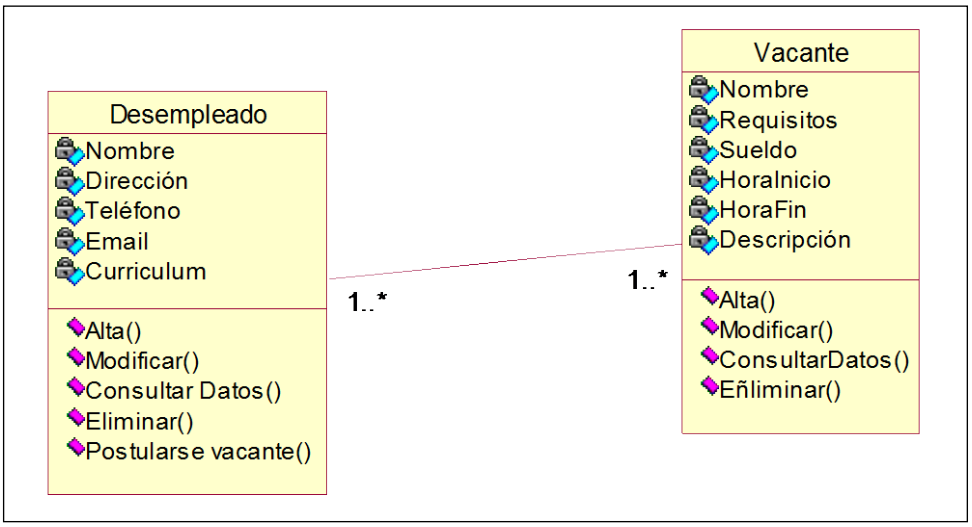


Figura 7. 4. Relación de asociación entre las clases Desempleado y Vacante. 1..* indica que uno o más Desempleados puedan postularse para una o más Vacantes.

  - **Agregación** es un tipo de asociación que denota que los objetos de una clase forman parte de un objeto de otra clase, o sea que una clase está *compuesta* por objetos de otras. Se denota por una línea con un rombo del lado de la clase compuesta. Ejemplo de este tipo de relación, es una computadora está *compuesta* por el CPU, monitor y teclado. En la figura 7.6 se muestra gráficamente esta relación.

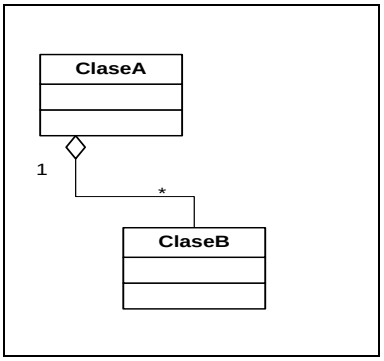


Figura 7. 5. Relación de agregación entre clases.

- **Generalización** relaciona a una clase general o abstracta que comparte sus atributos y operaciones con clases que los especializan. Las subclases heredan todos los atributos y operaciones de la superclase. La relación entre la clase general y sus especializaciones se denota por línea con un triángulo del lado de la general. Un ejemplo de esta relación es una figura geométrica que *puede ser* un triángulo, cuadrado, círculo, etc. A todas estas figuras se les puede medir el perímetro, el área pero cada una tiene su forma de calcularse. En la figura 7.7 se muestra gráficamente esta relación.

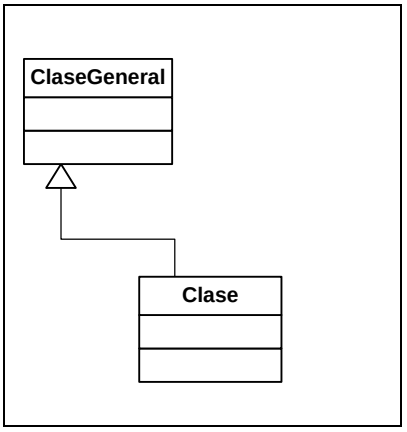


Figura 7. 6. Relación de generalización entre clases.

Identificación de clases

Para identificar las clases, se analiza cada caso de uso para imaginar qué clases se necesitan de cada estereotipo. Se recomienda iniciar por la identificación de las clases de control.

Por ejemplo, si se tiene un caso de uso de *Administrar Desempleado*, seguramente se necesitará una clase que represente a los desempleados en la aplicación. Entonces se propone la clase *Desempleado* como una clase de control. Los atributos de esta clase serán los datos necesarios de los desempleados y sus métodos pueden ser: alta, modificar, eliminar, consultar datos, etc.

Una técnica para identificar las clases son las *tarjetas CRC* [Beck, Cunningham]. Las tarjetas *CRC (Clase-Responsabilidad-Colaboración)* se usan para determinar tanto las clases como sus responsabilidades. Para cada responsabilidad se busca la colaboración con otras clases y así se identifican nuevas clases. Estas tarjetas tienen la forma mostrada en la figura 7.8.

Nombre de clase:	
Responsabilidad	**Colaboración**

Figura 7. 7. Forma de las tarjetas CRC.

Se llaman tarjetas porque inicialmente se usaban tarjetas de fichas bibliográficas, pero actualmente se usan tablas.

Clases de control

Para identificar estas clases se puede utilizar el siguiente proceso junto con la técnica de tarjetas de CRC:

1. Seleccionar un caso de uso y sus flujos de eventos.
2. Por cada paso de la interacción del actor con el sistema, se analizan los sustantivos más significativos como candidatos a clases que se requieren para realizar cada acción. Para cada clase candidata se escribe en una tarjeta su nombre, por ejemplo A. El nombre debe ser un sustantivo.
3. Cada acción asociada a la clase A, es una responsabilidad de esta clase y se anota en la columna *Responsabilidad*. La descripción de una responsabilidad deberá empezar con un verbo en infinitivo.
4. Se analiza la responsabilidad R y se trata de identificar qué otras clases deben colaborar con la clase A para poder cumplir con esa responsabilidad.
    a. En caso de identificar a las clases B y C como colaboradoras de esa responsabilidad se apuntan sus nombres en la columna de *Colaboración* de la clase A para la responsabilidad R. Para las clases B y C se repite el proceso de crear sus tarjetas y se anotan las responsabilidades de estas clases requeridas para la colaboración.
    b. En caso de no necesitar de colaboración, la columna de *Colaboración* se queda vacía.
5. Se regresa al punto 2 para analizar el siguiente paso de la interacción.
6. El proceso para un caso de uso termina cuando se finaliza la secuencia de pasos en sus flujos de eventos.
7. Se regresa al punto 1 para seleccionar al siguiente caso de uso.
8. El proceso termina cuando ya se analizaron todos los casos de uso.

Durante este proceso se van refinando y corrigiendo las tarjetas. Cuando ya se tiene un conjunto de tarjetas para la realización de todos los casos de uso, se construye el diagrama de clases a partir de éstas.

Para la construcción del diagrama de clases se puede utilizar nuevamente la técnica de tarjetas CRC:

1. Para cada tarjeta se crea una clase en el diagrama, con el mismo nombre, en singular y empezando con mayúscula.
2. Por cada responsabilidad de la clase, se define el método correspondiente, nombrado como verbo en infinitivo y en minúsculas.
3. Para cada colaboración entre clases, se dibuja una relación de asociación entre ambas clases.

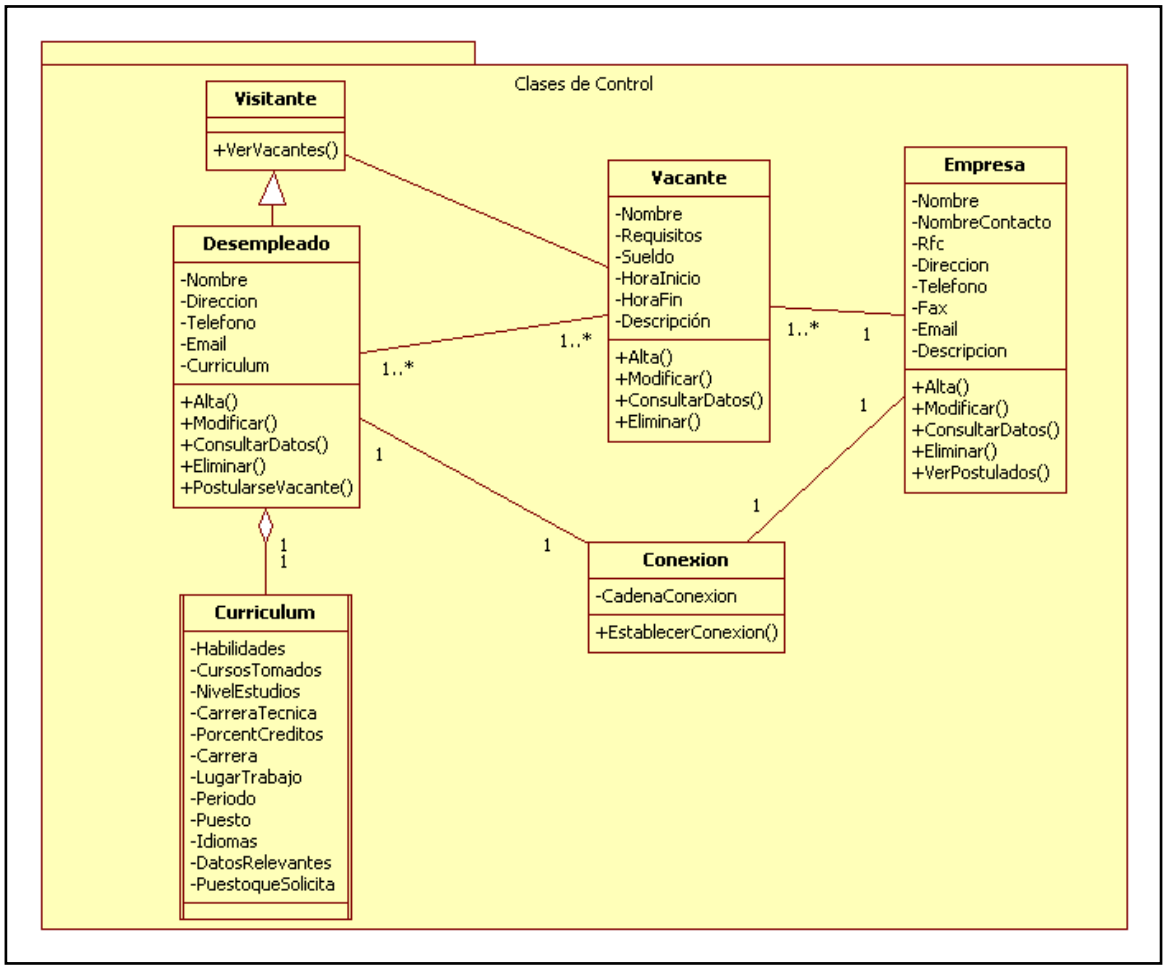


Figura 7. 8. Diagrama de clases de control del sistema de bolsa de trabajo.

En la figura 7.9 se muestra el diagrama de clases de control para el sistema de Bolsa de trabajo. La clase general Visitante se puede especializar en Desempleado por lo que esas clases tienen una relación de herencia. La clase Desempleado tiene una relación de agregación con la clase Currículum, porque cada Desempleado tiene un Currículum. La clase Vacante está asociada con las clases Empresa y Desempleado.

Es necesario remarcar, que este diagrama se irá detallando y modificando conforme se avanza en el diseño.

Al analizar todos los casos de uso, se puede construir uno o varios diagramas de clases de control. Si el sistema es muy complejo se construye un diagrama de clases para cada caso de uso. Si es simple, se puede construir un solo diagrama de clases de control para todos los casos de uso.

Clases de Interfaz

Una vez que se han diseñado las clases de control, se identifican las clases de interfaz de usuario. Para encontrarlas, se revisa el prototipo de interfaz de usuario creado en la fase de Especificación de requerimientos y se dibuja una clase para cada pantalla. Las clases de interfaz se podrán implementar con diversas tecnologías como serán ventanas, código html, jsp, etc. que se decidirá posteriormente.

En la figura 7.10 se hace una diferencia en los estereotipos en las clases de interfaz. Se usa el estereotipo <<Pantalla>> para las clases que sirven para mostrar la información, el estereotipo <<Forma de entrada>> cuando será una clase de captura.

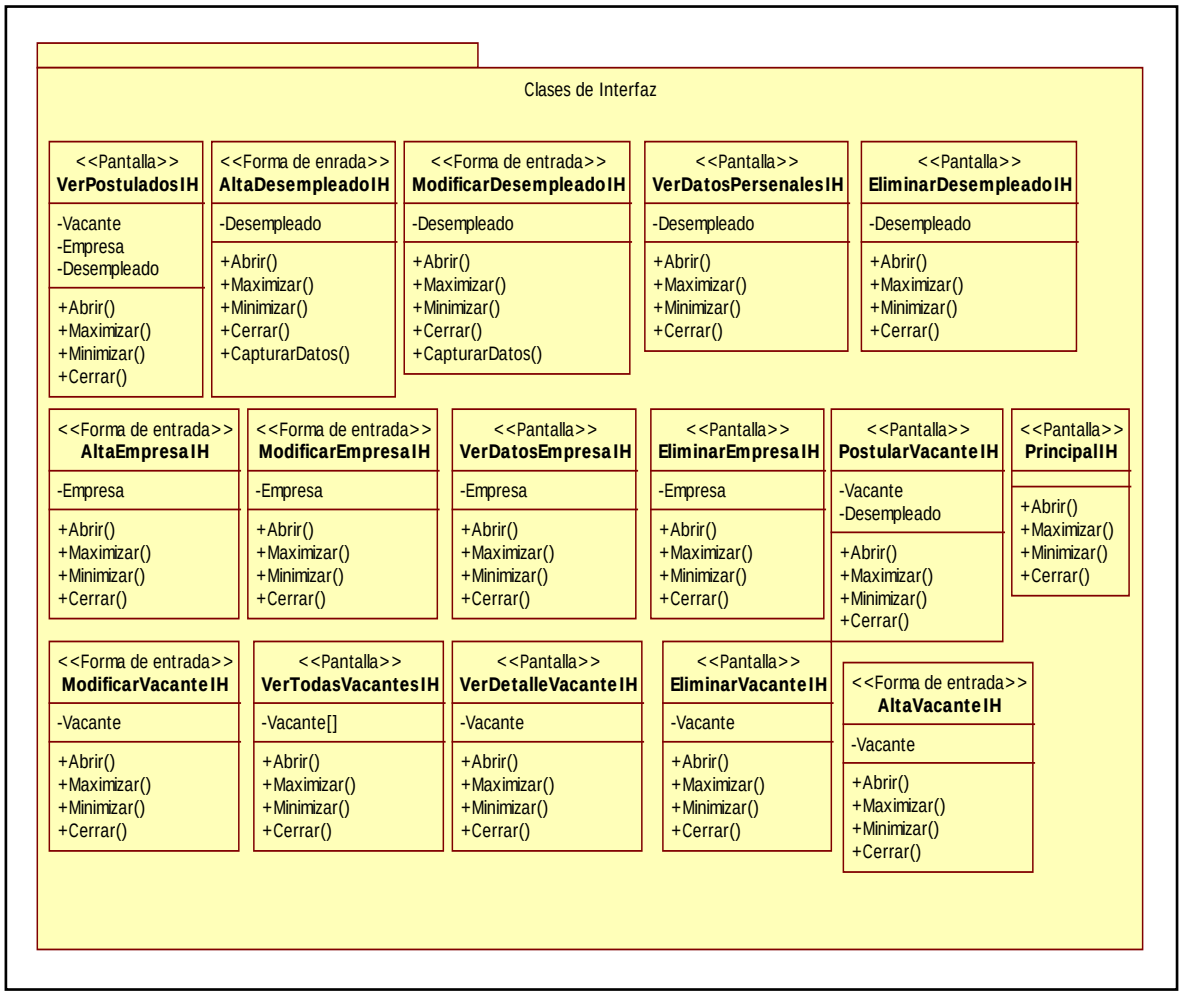


Figura 7. 9. Diagrama de clases de interfaz del sistema de bolsa de trabajo.

Clases de Entidad

Para identificar las clases de Entidad, se escogen las clases de control cuyos atributos deben ser guardados. Para cada una de estas clases, se dibuja una clase de tipo entidad, que se encargue de resguardar estos atributos en una base de datos o un archivo. Las clases de Entidad para el sistema de bolsa de trabajo se muestran en la figura 7.11.

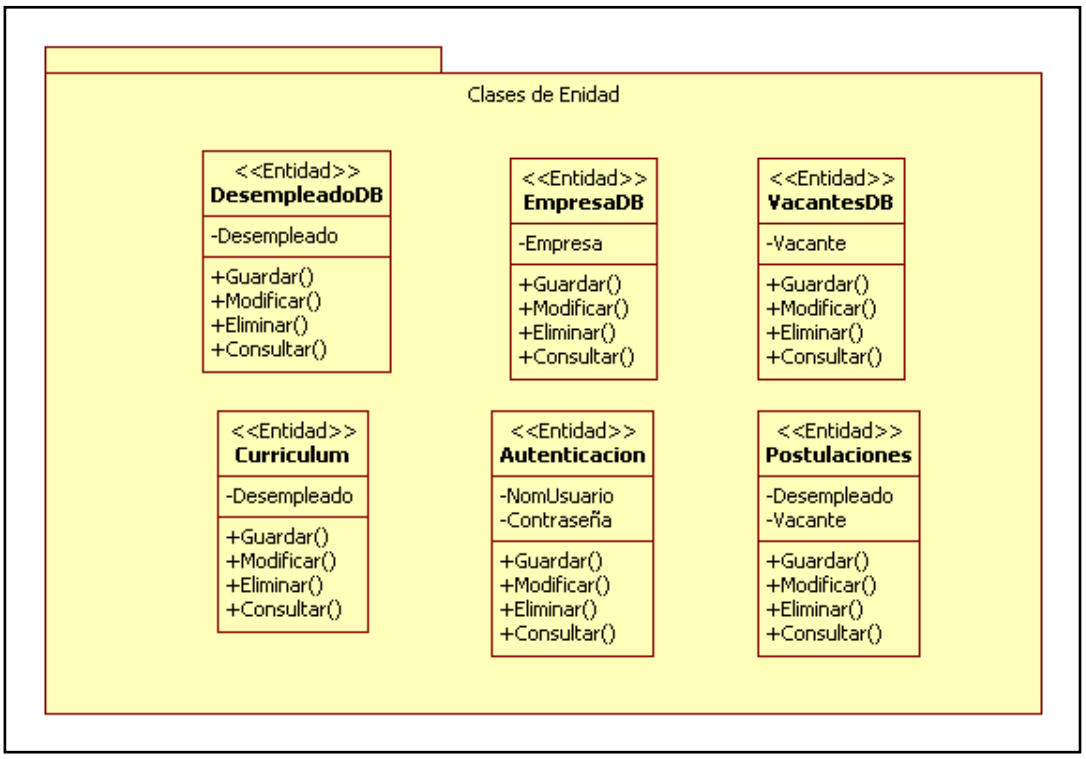


Figura 7. 10. Diagrama de clases de entidad del sistema de bolsa de trabajo.

7.4.2 Vista dinámica

La vista dinámica describe cómo interactúan los objetos para entregar la funcionalidad requerida del sistema. Hay varios diagramas de UML para representar esta vista. Los diagramas de secuencia describen la interacción entre los objetos de las clases que intervienen en cada caso de uso. Los diagramas de estado modelan el cambio de estados de entidades del sistema.

Diagramas de secuencia

Los diagramas de secuencia muestran la interacción entre los objetos de las clases como una secuencia de envío de mensajes entre ellos, ordenados en el tiempo. Los elementos de estos diagramas son:

- **Actor** es el iniciador de la primera invocación del método la secuencia de mensajes que se envían los objetos entre sí. En la figura 7.14 se muestra un ejemplo.

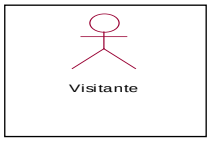


Figura 7. 11. Actor Visitante del sistema de bolsa de trabajo.

- **Objetos** son instancias de las clases. Se representa por un rectángulo con el nombre del objeto y la clase a la pertenece subrayados. En el diagrama, cada objeto tiene una línea vertical que representa su línea de tiempo que avanza de arriba – abajo. En la figura 7.12 se muestra un ejemplo.

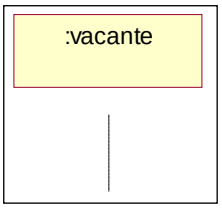


Figura 7. 12.Objeto de la clase Vacante en un diagrama de secuencia, para el sistema de bolsa de trabajo.

- **Mensajes** que son enviados de un objeto, que es el fuente, a otro, el receptor, a través del tiempo. Representa la invocación del método del objeto receptor. Se modela como una línea con la flecha del objeto fuente al receptor con el nombre del método sobre la línea el cual puede o no contener los parámetros del método. Se muestra un ejemplo en la figura 7.13.

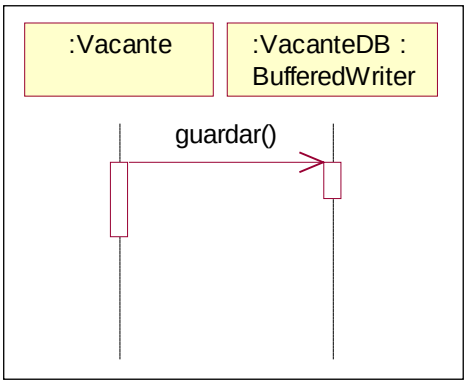


Figura 7. 13. Envío del mensaje "guardar del objeto de la clase Vacante a un objeto de la clase VacanteDB receptor en el sistema de bolsa de trabajo.

En un diagrama de secuencia se coloca arriba a la izquierda al actor o al objeto que inicia la interacción entre objetos y arriba hacia la derecha se ponen todos los objetos que participan en la interacción. Los mensajes que los objetos se envían se dibujan con flechas entre las líneas de vida de los objetos, etiquetados con los métodos. Los mensajes fluyen de izquierda a derecha y de arriba hacia abajo representando el orden en el tiempo.

Diagramas de secuencia para los casos de uso

Para cada flujo normal y excepcional de eventos en los casos de uso, se construyen un diagrama de secuencia. Estos diagramas representan la vista dinámica de los casos de uso especificados en los requerimientos. Para su construcción se parte del detalle de los casos de uso.

Por cada flujo de un caso de uso se identifican las clases de cada capa necesarias para su realización y se crea un diagrama de secuencia de la siguiente manera:

1. Se representa al actor que corresponde al caso de uso poniéndolo arriba en el extremo izquierdo del diagrama.
2. El actor inicia las acciones del caso de uso enviando un mensaje a un objeto de una clase de interfaz identificada para este flujo.
3. Enseguida se coloca uno o más objetos de clases de control y si es necesario, uno o más objetos de entidad.
4. Cada mensaje entre objetos aparece como una flecha dirigida del objeto fuente al objeto receptor, etiquetado con el nombre del método correspondiente.
5. Para visualizar el orden temporal del envío de los mensajes, la flecha de un mensaje posterior se dibuja un poco más abajo que la del mensaje anterior.

Específicamente sobre los diagramas de secuencia para los flujos de los casos de uso, se pueden aplicar ciertas reglas [Rosenberg]:

1. Los actores pueden enviar mensajes solo a los objetos de las clases de interfaz.
2. Los objetos de las clases de interfaz solo pueden enviar mensajes a los objetos de las clases de control.
3. Los objetos de las clases de entidad pueden enviar mensajes solo a los objetos de las clases de control.
4. Los objetos de las clases de control pueden enviar mensajes a objetos de las clases de interfaz, a los de las clases de entidad y a otros objetos de las clases de control, pero no a los de actores.

En la figura 7.15, se muestra un diagrama de secuencia para el caso de uso de *Consultar Vacante*, en que el *Visitante* (1) abre un objeto de la clase de interfaz de *VerTodasVacantesIH*, (2) consulta los datos del objeto de la clase *Vacante*, quien (3) los extrae de un objeto de la clase entidad *VacanteDB*, se los entrega al objeto de *Vacante* (4), quien los pasa a un objeto de la clase de interfaz de *VerTodasVacantes* (5) para que los vea

el visitante. El actor selecciona una vacante (6) de la interfaz y se muestra su detalle en un objeto de la clase de interfaz de *VerDetalleVacante* (7).

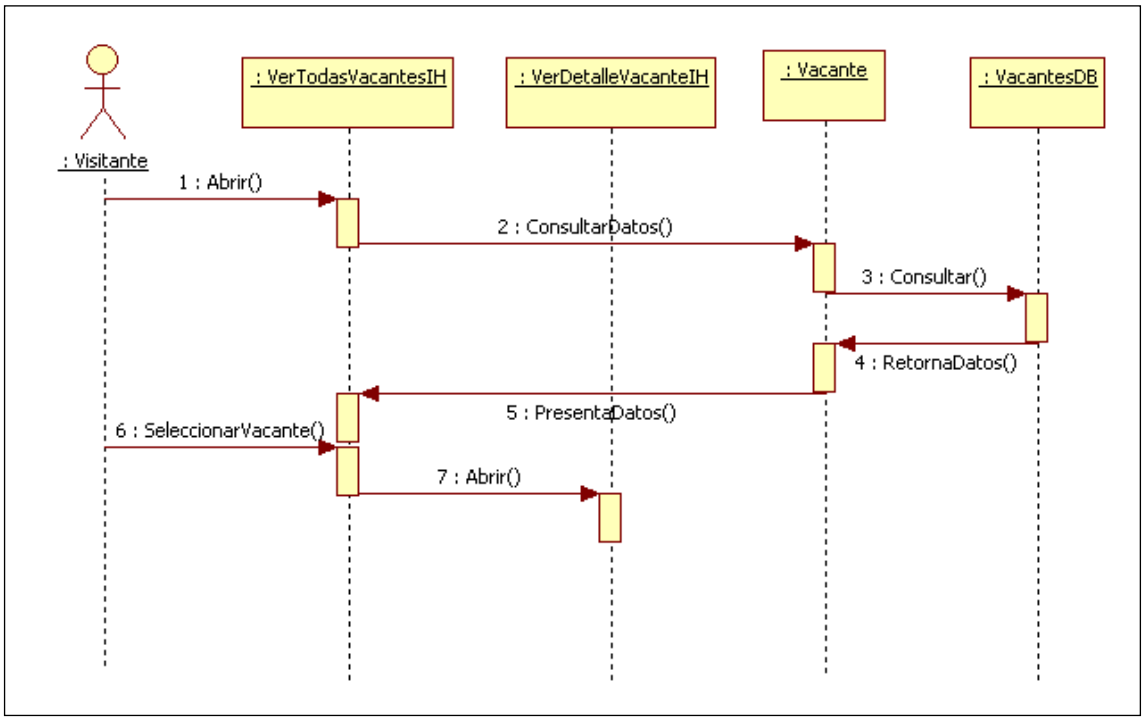


Figura 7. 14. Diagrama de secuencia del flujo normal del caso de uso Consultar Vacantes del sistema de bolsa de trabajo.

Es común que al hacer los diagramas de secuencia surjan nuevas clases y métodos no identificados durante la construcción de los diagramas de clase. Por lo tanto, una vez terminados los diagramas de secuencia para todos los casos de uso, se revisa la consistencia entre ambos los diagramas de clase y de secuencia. Se modifican los diagramas de clases según lo encontrado durante la construcción de los diagramas de secuencia.

Los cambios en los diagramas, significan que se está entendiendo mejor el problema y el proceso se avanza de manera iterativa en la construcción de la solución computacional.

Diagramas de estados

Para modelar el cambio de estados de entidades del sistema, que es otro aspecto de la vista dinámica, se usan los diagramas de estados de UML.

Un diagrama de estados modela una máquina de estados finitos o autómata, que enfatiza el flujo de control de un estado a otro. Es una gráfica con nodos que representan estados y por arcos dirigidos que representan transiciones entre los estados a causa de eventos. Los elementos de estos diagramas son:

- Un **estado** es la condición de una entidad del sistema. Puede caracterizar a un objeto o representar a una pantalla, que cambian de estado a causa de eventos. Los estados se representan por rectángulos con esquinas redondeadas con el nombre del estado. Se muestra un ejemplo en la figura 7.16.

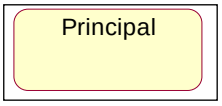


Figura 7. 15. Estado que representa la pantalla principal del sistema de bolsa de trabajo.

- Un **evento** es algo significativo que ocurre en un momento y que causa el cambio de estado.

- Una **transición** modela el cambio de estado a causa de un evento. Se representa por una flecha que va de un estado a otro. La flecha se puede etiquetar con el nombre del evento. En la figura 7.17 se muestra su representación gráfica.

Figura 7. 16. Representación gráfica de una transición.

- El **estado inicial** se dibuja como un círculo de color negro.

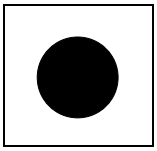


Figura 7. 17. Representación gráfica del estado inicial.

- El **estado final** que se denota por dos círculos concéntricos sobrepuestos, el del centro de color negro. Puede haber varios estados finales o ninguno. Su representación gráfica se muestra en la figura 7.19

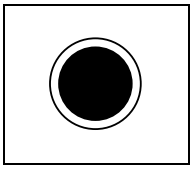


Figura 7. 18. Representación gráfica del estado final.

Diagramas de estado para la navegación

Para modelar la navegación en la interfaz de usuario, que es un aspecto dinámico de la construcción del sistema de software, se usan diagramas de estados. La navegación en la interfaz de usuario es el cambio de pantallas que ve el usuario a causa de eventos que cambian el estado del sistema, como por ejemplo, la selección de una opción en un menú cambia el estado y puede aparecer otra pantalla.

Para construir el diagrama de estados, que modela la navegación, se parte del prototipo de interfaz de usuario construido en la fase de Especificación de requerimientos. Las pantallas se representan por estados y los eventos, que ocasionan el cambio de un estado a otro, por las transiciones entre estados.

Para modelar la navegación en la interfaz de usuario con diagramas de estado, el estado inicial del diagrama, apunta al estado que representa a la pantalla de principal del sistema. En esta pantalla, por lo general, se tienen varias opciones para navegar. Cada opción representa a un evento que puede llevar a otra pantalla. En el diagrama, cada pantalla se representa por un estado y los eventos etiquetan a las transiciones entre los estados.

En la figura 7.20 se muestra el diagrama de estados para el sistema de bolsa de trabajo. El estado inicial representa a la pantalla *Principal*. Dependiendo de las opciones del menú que se elijan, se pasa a los estados correspondientes. Si en el estado *Principal*, el evento es que se tiene un *Registro Erróneo*, la transición lleva al mismo estado.

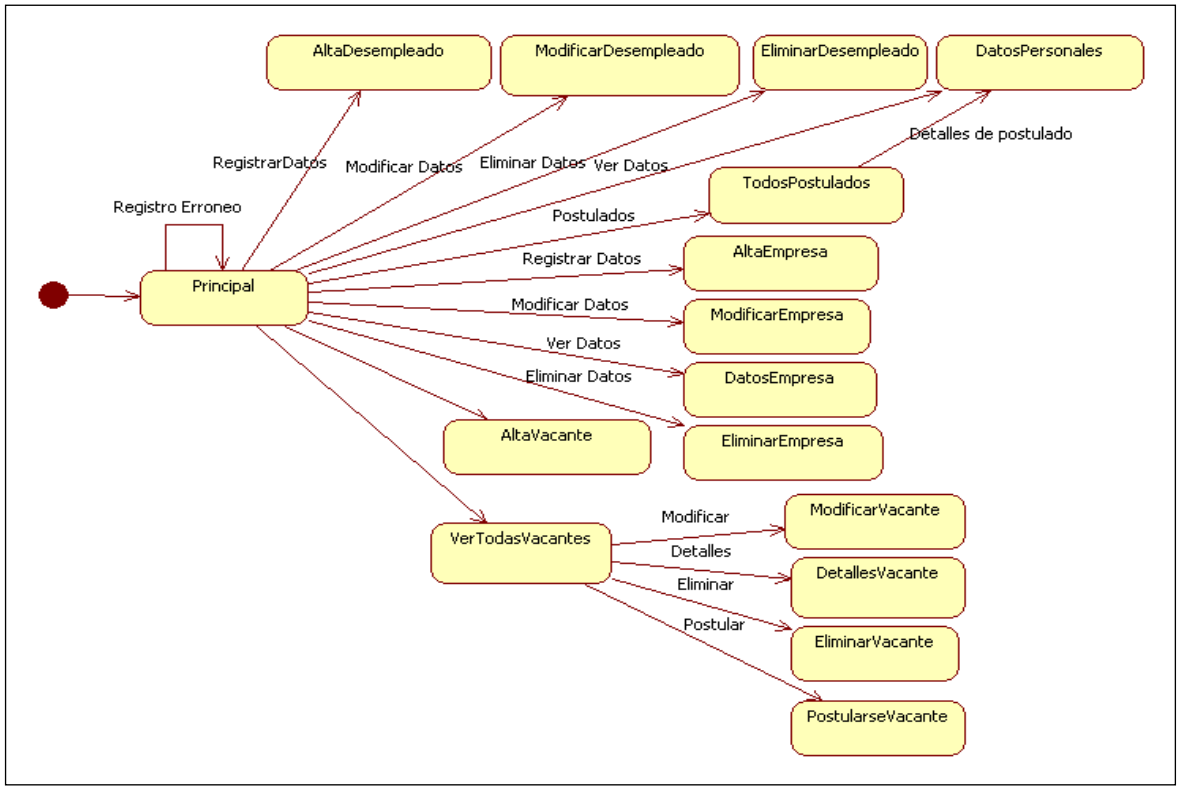


Figura 7. 19. Diagrama de estados para la navegación del sistema de bolsa de trabajo.

Otro uso de los diagramas de estado, es modelar el orden válido de ejecución de casos de uso. Por ejemplo, el caso de uso de "*Autentificar al usuario*" que debe realizarse antes de cualquier otro caso de uso. Para construir este modelo, se parte del diagrama general de casos de uso. Cada caso de uso se modela como un estado y el paso de un caso de uso a otro a consecuencia de un evento, se modela por una transición.

Diseño de la base de datos

A partir de las clases de tipo persistente de la capa de almacenamiento, se diseña la base de datos. Para modelar la base de datos se usan los conocimientos y técnicas de un curso de Bases de Datos.

Uno de los diagramas más utilizados en este tema, es el de Entidad-Relación, en el cual se hace corresponder cada clase de control del diagrama de clases, que se considere persistente, a una Entidad del diagrama Entidad-Relación. Los atributos de las clases se convierten en los atributos de las Entidades. Las relaciones entre las Entidades se construyen a partir de las relaciones entre clases.

7.5 Plan de pruebas de integración

Cuando ya se tienen todos los elementos de los paquetes de la arquitectura diseñados, se planea como se irán integrando en un sistema completo.

El *Plan de pruebas de integración* describe el orden en que se juntan los elementos de los paquetes de la arquitectura. Además, se proponen las pruebas que se aplicarán para verificar la correcta interacción entre estos elementos. Este plan ayuda a revisar y completar el diseño arquitectónico.

Estrategias para la integración del sistema

Para definir el orden de integración de los elementos del sistema, se puede seguir alguna de estas estrategias [Binder]:

- **Estrategia de paquetes.** La idea es hacer las integraciones según los paquetes del diseño. Por ejemplo, se integran todos los elementos del paquete de interfaz de usuario para un actor. Este enfoque tiene la ventaja, de que se integrarán los elementos de un paquete y luego se conjuntan con los integrados de otro paquete siguiendo la arquitectura.

- **Estrategia por funcionalidad o casos de uso.** La idea es integrar los elementos de los paquetes de las tres capas que corresponden a la realización de cada caso de uso. Se puede planear la integración de casos de uso en paralelo cuando no hay dependencia entre ellos.

No existe una estrategia adecuada para todos los sistemas, definir cuál usar es decisión del equipo, el proyecto y el nivel de diseño efectuado.

La integración empieza con los elementos que no dependa de otros. En el caso de las clases, una clase depende de otra si invoca alguno de sus métodos. Bajo esta regla, se puede crear una gráfica de dependencias entre clases que sirve de guía para definir el orden de integración.

En el Plan de pruebas de integración se incluye los métodos que se deberán probar al hacer la integración de las clases.

PLAN DE PRUEBAS DE INTEGRACIÓN:
Sistema de bolsa de trabajo

La estrategia a seguir para el plan de pruebas de integración será la de Dependencia de clases, la cual estará basada en la dependencia estática de las clases, definida en el diagrama de clases.

El orden de las clases que se implementarán es:

1. Conexion.java
2. ConsultaBD.java
3. DesempleadoDB.java, EmpresaDB.java, VacantesDB.java, Curriculum.java,
4. Autenticacion.java y Postulaciones.java. Se pueden implementar paralelamente.
5. Visitante.java, Vacante.java, Empresa.java y Vacante.java. Se pueden implementar paralelamente.
6. PrincipalIH.java, AltaEmpresaIH.java, AltaVacanteIH.java,
7. VerTodasVacantesIH.java y VerDetalleVacanteIH.java. Se deben implementar en ese orden.

El orden de implementación será estricto, excepto para las clases que se pueden implementar en paralelo; esto es para asegurar que no se utilice tiempo extra en probar las clases, es decir, para probar una clase se deberán tener implementadas las clases de las cuales depende.

El tipo de pruebas que se realizarán serán pruebas que permitan mostrar la correcta funcionalidad de las clases, y subsecuentemente se realizaran pruebas para corroborar que las clases dependientes funcionan correctamente al utilizar la referencia a las clases de las que dependen, mostrando con esto que las clases se integran de manera uniforme y funcional.

Figura 7. 20 Ejemplo del Plan de pruebas de integración para el sistema de bolsa de trabajo.

Este plan servirá para hacer la integración al ir terminando la construcción de los elementos de los paquetes.

7.6 Profundización de temas para el segundo ciclo

7.6.1 Especificación de Diseño de software

La especificación del diseño debe ser clara y precisa, por lo que incluye varios diagramas de UML como son: diagramas de paquetes, diagramas de clases, diagramas de secuencia y si fuera necesarios diagramas de estados y de distribución. Se construye el documento de especificación del diseño que según estándares internacionales y Moprosoft debe incluir:

1. Arquitectura general del sistema. Contiene la estructura interna del sistema, es decir la descomposición en componentes, y las relaciones entre ellos.
    a. Diagrama de paquetes o componentes
2. Detalles de las clases por componente y cómo interactúan en cada caso de uso
    a. Diagramas de clases por componente
    b. Diagramas de secuencia por cada caso de uso del componente
3. Distribución de los componentes en los nodos si el sistema es distribuido
    a. Diagrama de distribución
    b. Si es necesario se hacen diagramas de navegación entre los componentes con diagramas de estados.
4. Plan de pruebas de integración.

7.6.2 Diseño para el reuso

Cuando los miembros del equipo hacen sus diseños es importante que obedezcan los estándares de diseño fijados por todos que facilitará la integración y también favorecer la reutilización de componentes en iteraciones posteriores acortando el tiempo de desarrollo que incrementando la productividad del equipo.

Para que la reutilización sea factible, se deberá probar a conciencia cada componente, a fin de poder confiar en él al incorporarlo en otros desarrollos.

Una tarea para promover la reutilización tanto para el segundo ciclo del curso como en empresas desarrolladoras de software es la identificación de componentes con funcionalidades comunes útiles. La identificación de componentes reusables se debe planear desde la fase de estrategia y de requerimientos.

Un aspecto importante de la reusabilidad es construir componentes que estén autocontenidos y suficientemente aislados (alto grado de cohesión y bajo de acoplamiento). Otro aspecto importante es la definición de la comunicación entre componentes a través de la definición de un estándar para el paso de parámetros que elimine confusiones y reduzca errores. La definición del estándar para paso de parámetros incluye la definición de los

nombres para los parámetros de entrada, de salida, los mensajes y condiciones de error. Además se especifica la forma de tratar los errores (uso de excepciones).

Para facilitar el conocimiento de los componentes reusables disponibles para el equipo, el administrador de apoyo o el de desarrollo, debería mantener un catálogo de componentes.

El catálogo contiene información simple que realmente promueva en los desarrolladores el reuso. Podrían ser tablas (una para cada categoría de componentes, como clases de interfaz o de control, etc.) con la siguiente información:

- Nombre del componente y última versión.
- Autores.
- Breve especificación de su funcionalidad con requerimientos para llamarlo y tipo de salidas que produce.
- Localización en el depósito del equipo.

Otro elemento que indispensable para el reuso es la buena calidad de los componentes. Cada componente debe estar probado, contener buenos comentarios, ser claro, preciso y completo. Para asegurar esta calidad de los componentes son esenciales las reuniones de revisión en el diseño y codificación de los componentes, así como las pruebas unitarias que aseguren que funcionen adecuadamente.

Algunos problemas asociados al reuso son: el mantenimiento de ese catálogo, de los componentes en sí, encontrar y adaptar los componentes. Para resolverlos debería contarse con herramientas que faciliten el manejo de estos catálogos.

Al iniciar ciclos de desarrollo posteriores al primero, se debe motivar al equipo a reusar lo ya elaborado ya que esto disminuye el tiempo de desarrollo y aumenta la calidad si los componentes son de calidad. Pero esto implica mantener la lista de componentes reusables que es una tarea del administrador de apoyo en el segundo ciclo. Esto incluye colectar los componentes y darlos a conocer al equipo.

Referencias bibliográficas

- Alexander C., Ishikawa S., Silverstein M., Jacobson M., Fiksdahal-King I., Angel S. "*A Pattern Languaje. Towns.Building.Construction*". Oxford University Press 1977.
- Arlow J., Neustadt I. *UML2 and the Unified Process. Practical Object-Oriented Analysis and Design.* 2a edición. Addison Wesley 2005.
- Beck K., Cunningham W., SIGPLAN Notices October 1989, vol. 24 (10).
- Booch G., Rumbaugh J., Jacobson I. "*The Unified Modeling Languaje. Users guide*". Addison Wesley 1999.
- Braude E. J. "*Ingeniería de software. Una perspectiva orientada a objetos*". Alfaomega 2003.

- Buschmann F., Meunier R., Rohnert H., Sommerland P., Stal P. *"A system of patterns"* John Wiley 1996.
- Coad P. at. al. "*Object Models, Strategies, Patterns and Applications"*. Yourdon Press Computing Series, Prentice Hall. 1995
- Eeles P., Houston K., Kozaczynski W. *"Building J2EE Applications with the Rational Unified Process"*. Addison Wesley 2003.
- Humphrey Watts S., "*Introduction to Team Software Process*", SEI Series in Software Engineering, Addison Wesley, 2000.
- Jacobson I., Booch G., Rumbaugh J. "*The* U*nified Software Development Process*". Addison Wesley 1999.
- Rosenberg Doug, Scott Kendall. *"Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example"*. Addison Wesley 2001.
- Tanenbaun A. S. "*Modern operating systems*". Prentice Hall 1992

Capítulo 8

Fase de Construcción

Objetivos del capítulo:

Describir las actividades necesarias en la fase de construcción tales como: hacer el diseño detallado, construir el código, planear y aplicar pruebas unitarias.

8.1 Fase de Construcción: objetivos, actividades y productos.

El objetivo de esta fase es hacer la construcción del software. Las actividades de diseño detallado, programación y pruebas unitarias, son parte de esta fase.

La construcción de software es un proceso iterativo. Parte del diseño de la fase anterior, se refinan sus componentes introduciendo elementos del lenguaje de programación. Este refinamiento se repite hasta que se pueda derivar el código.

Para hacer las tareas de la fase de construcción, el equipo completo revisa los modelos que se crearon en la fase de diseño. Se reparten los componentes a los ingenieros de desarrollo y se fija fecha para la entrega de código ya probado.

Cada ingeniero de desarrollo detalla los diagramas de clases de los componentes que le corresponden, genera el código y lo prueba. Estas actividades son iterativas y se repiten hasta que el ingeniero esté satisfecho con sus productos.

Objetivo:

O1. Hacer el diseño detallado
Se realiza el diseño detallado mapeando las clases del diseño a clases y paquetes en el ambiente del lenguaje de programación.

Actividades:

A1.1 Refinar los diagramas de clases hasta el nivel necesario para hacer la codificación.

Productos:

- Diagramas detallados de clases incluyendo: los nombres de todos los atributos y sus tipos; métodos con sus parámetros y su resultado; y seudo-código para métodos complejos.

Objetivo:

O2. Construir el código
Se construye el código para las clases de los paquetes en el ambiente de programación, a partir de los diagramas detallados de clases.

Actividades:

A2.1 Codificar las clases y paquetes siguiendo el estándar de codificación.

Productos:

- Código para las clases y paquetes.

Objetivo:

O3. Planear y efectuar las pruebas unitarias
Para cada clase que se hace un plan para probarla. Se planean los tipos de prueba y se definen los casos de prueba que deberán aplicarse. Los tipos de prueba a aplicar son llamadas de cajas blancas y negras.

Actividades:

A3.1 Hacer el Plan de pruebas unitarias.
A3.2 Hacer pruebas unitarias de caja blanca, registrar defectos y corregirlos
A3.2 Hacer pruebas unitarias de caja negra, registrar defectos y corregirlos.

Productos:

- Plan de prueba unitaria de cada clase.

Formas a entregar en todas las fases:

- Semana personal
- Semana del equipo
- Minuta de la reunión
- Lista de verificación
- Registro de defectos

Resumen de productos a entregar:

Producto	Responsable
Diagramas detallados de clases	Ingeniero de desarrollo
Código para las clases y paquetes	Ingeniero de desarrollo
Plan de prueba unitaria de cada clase	Ingeniero de desarrollo

8.2 Diseño detallado de clases

En el diseño detallado se completan los diagramas de clases incluyendo los detalles necesarios para su codificación. El nivel al que se detallan estos diagramas debe ser suficiente para que cada ingeniero construya su código, pero también debe proporcionar la información necesaria para que otros ingenieros construyan el suyo.

El detalle de las clases de control incluye: nombres y tipos de todos los atributos, para los métodos se especifican los parámetros con sus tipos y el tipo del valor de retorno. Si algún método requiere para su implementación de un algoritmo más complejo, éste se especifica en seudo-código. La figura 8. 1 muestra un ejemplo de diseño detallado de una clase

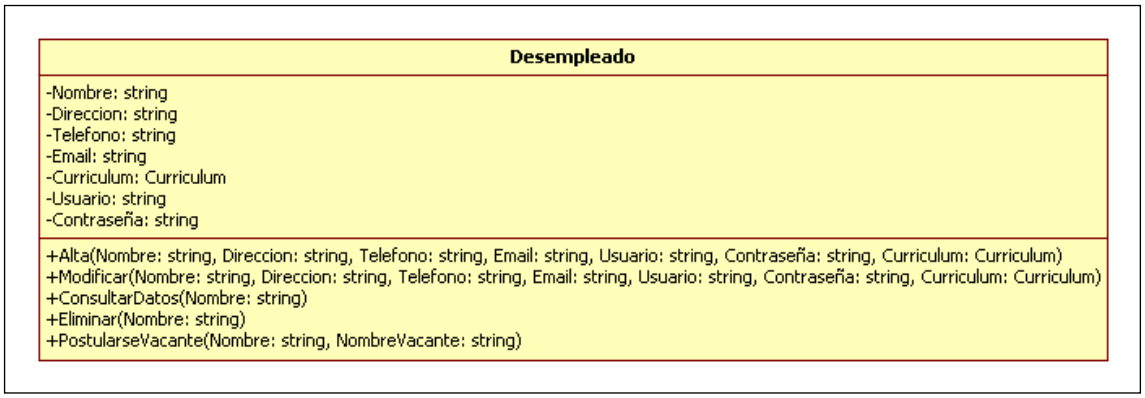


Figura 8. 1. Ejemplo del diseño detallado de una clase.

Cada ingeniero de desarrollo realiza el detalle de los componentes que son su responsabilidad. De ahí la importancia de seguir el estándar de documentación de diseño establecido en la fase anterior.

No todas las clases se codifican en un lenguaje de programación. Por ejemplo, en una aplicación web, las clases de interfaz pueden implementarse como código html. Por otro lado, a partir de las clases de entidad, se diseña la base de datos.

En los paquetes se incluyen las clases necesarias del ambiente de programación o las bibliotecas disponibles para la construcción de las clases.

Algunas herramientas de diagramación de UML ayudan a completar el diseño detallado a partir de los diagramas de clases, generando un esqueleto de código. Cuando se tienen dichas herramientas la tarea de codificar consiste en detallar los diagramas y completar los esqueletos con el código. Otra ventaja de estas herramientas es que se mantiene la integridad entre los diagramas y el código, por lo que los cambios en uno se reflejan en los otros. En la figura 8. 2 se muestra un ejemplo de un diagrama de clases detallado.

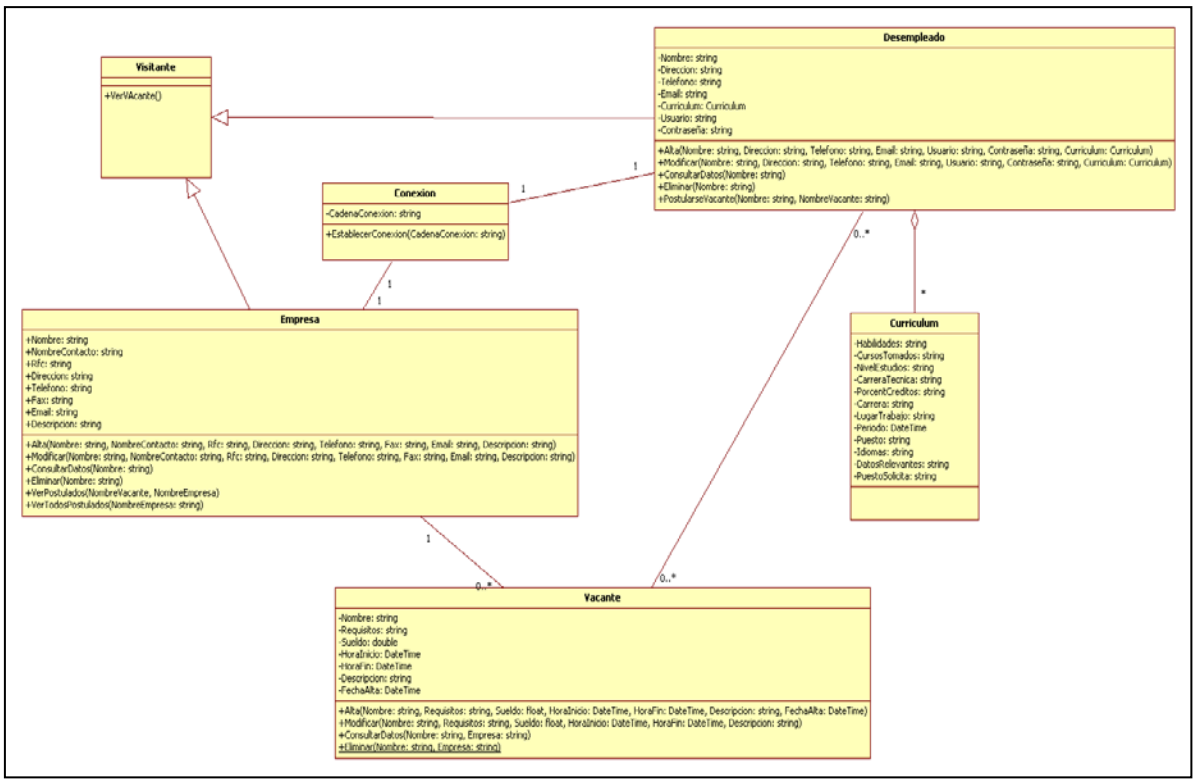


Figura 8. 2. Diagrama de clases detalladas.

8.3 Construcción de Código

Estándares de Codificación

El código de un lenguaje de programación se genera con el fin de que sea compilado y ejecutado por la máquina. Sin embargo, este mismo código también tiene que ser comprendido por los seres humanos. En primer lugar debe de entenderlo el propio autor del programa aunque pase un mes desde que lo creó por primera vez. También deben de entenderlo otros desarrolladores, que forman parte del equipo, o gente que dará el mantenimiento al programa. Con tal objetivo se establece un *estándar de codificación*, además del de documentación de diseño.

Un estándar de codificación es un conjunto de normas para hacer los programas más legibles. Define las convenciones para escribir los encabezados de los paquetes o clases, para nombrar clases, atributos, métodos y parámetros, para estructurar de manera legible las instrucciones de control, manejo de errores etc.

Por ejemplo, un estándar de comentario de encabezado de una clase puede pedir que éste contenga los siguientes elementos: el propósito de la clase, sus autores, la fecha de creación, su versión actual y referencias cruzadas a otras clases o archivos.

Se recomienda usar estándares de codificación ya existentes para el lenguaje de programación que se va a usar. Por ejemplo, para el lenguaje de programación Java se puede usar el estándar definido en el sitio de Java de Sun Corporation. En su defecto, se sugiere generar el estándar propio acordado por el equipo de desarrollo y documentarlo.

Revisión de código y programación entre pares

Para evitar defectos en el código que se genera a partir del diseño detallado, [Humphrey] propone leer el código antes de la compilación para eliminar defectos de forma temprana. Es una actividad que es poco practicada por las prisas de los ingenieros para que el compilador encuentre los defectos en el código recién escrito. Pero al compilador se le escapan los defectos de la lógica del programa. La lectura del código por una persona puede detectar esa clase de defectos y corregirlos de manera oportuna.

Una alternativa para generar el código y leerlo a la vez por dos personas, es la *programación entre pares* [Beck]. La idea principal es que el código se escribe entre dos personas sentadas frente de una sola máquina. Un miembro de la pareja tiene el control sobre el teclado y el ratón y es el que está codificando. La otra persona lee lo que se va escribiendo, analiza el código, revisa la lógica y la sintaxis, y comenta posibles alternativas y errores a su pareja. Este trabajo es dinámico, los roles dentro de la pareja pueden cambiar constantemente.

Con esta práctica se hacen revisiones constantes al código pues quien no tiene el control del teclado, está leyéndolo en cuanto se teclea y detecta errores que sería difícil y tardado de encontrar posteriormente.

Aunque aparentemente con la programación entre pares se avanza mas lento, la práctica ha demostrado que el código que se obtiene de esta manera, tiene mucho menos errores que el producido tradicionalmente, por lo que tiene una mayor calidad.

8.4 Pruebas unitarias

Las pruebas unitarias consisten en probar cada unidad lógica de código que se va terminando. En la orientación a objetos, la unidad lógica es la clase. Cada ingeniero deberá asegurar la calidad de sus clases probándolas detenidamente.

Las clases se prueban a través de la invocación de sus métodos. Para probar los métodos de una clase se definen *casos de prueba.* Un caso de prueba de un método consiste en definir un conjunto representativo de valores para los parámetros del método y el valor del resultado esperado para ese conjunto. Para cada método se pueden definir uno o más casos de prueba según la complejidad del método.

Plan de pruebas unitarias

Para hacer las pruebas unitarias de clases cada ingeniero define su *Plan de pruebas unitarias* para las clases que está construyendo.

En el Plan de pruebas unitarias se define: las clases que se probarán, los métodos y los casos de prueba para cada método. Una forma de especificar este plan es hacer una tabla con 4 columnas:

- La clase que se probará.
- Método a probar.
- Los casos de prueba con los conjuntos de valores para los parámetros para cada método.
- El resultado esperado de cada caso de prueba.

En la figura 8. 3 se muestra un ejemplo de definición de casos de prueba para los métodos de una clase.

Clase	Método	Caso de Prueba	Resultado Esperado
UsuarioRegistrado	setNombre(nombre)	miNombre	Guarda el nombre
UsuarioRegistrado	setDireccion(calle,colonia, delegación, CP)	miCalle 1 miColonia 1 MiDelegación 12345	Guarda los campos de la dirección

Figura 8. 3. Ejemplo de Plan de pruebas unitarias.

Realización de las pruebas unitarias según el Plan de pruebas unitarias

Para realizar la prueba unitaria de una clase, se crea un objeto de esa clase. Se invoca cada método a probar con los parámetros definidos en sus casos de prueba. Si el resultado obtenido coincide con el esperado, el método pasa esa prueba. Si no coincide, se revisa el código del método para encontrar la causa del defecto y corregirlo. Se vuelve a aplicar el mismo caso de prueba hasta que se obtengan el resultado esperado. Cuando una clase aprueba todos los casos de pruebas definidos en el Plan de pruebas unitarias, es aceptada.

8.4.1. Técnicas para definir los casos de prueba.

Para definir los casos de prueba se usan dos técnicas: las de *caja blanca (transparente)* y las de *caja negra*. En las pruebas de caja blanca se toma en cuenta la estructura del código de un método y se busca que durante las pruebas cada instrucción se ejecute al menos una vez. Las de caja negra, consideran al código del método como un todo oculto, verificando que para cada conjunto de valores de los parámetros se obtenga el resultado esperado.

A continuación se explican en qué consisten ambos tipos de prueba y algunas técnicas para planear ese tipo de pruebas.

Pruebas de caja blanca

Las pruebas de caja blanca también se llaman *pruebas estructurales.* Se revisa la estructura lógica de la unidad a probar tratando de definir casos de prueba que permitan ejecutar por lo menos una vez, cada una de las instrucciones del método.

Una técnica que se usa como guía para saber cuántos casos de prueba se deben definir para recorrer todas las instrucciones de la unidad por lo menos una vez, es la *complejidad ciclomática de McCabe.* Esta complejidad ciclomática se define como el número de condiciones más 1. Una condición es un punto de decisión en el código como por ejemplo *if, while, for,* etc.

Para planear las pruebas de caja blanca usando la complejidad ciclomática se determina el número de casos de prueba necesarios usando la fórmula de la complejidad ciclomática. Según este número, se definen los casos de prueba como conjuntos de valores de las variables, que permitan recorrer las diferentes trayectorias del código. Para cada condición, se eligen valores de las variables para que sea en una ocasión verdadera y en otra falsa. Para cada conjunto de valores se define el resultado esperado.

Ejemplo de un plan de pruebas con la complejidad ciclomática:

En la figura 8. 4, se muestra un pseudocódigo de un método que revisa si un alumno ya tiene calificación para un curso *c1*. Se tiene una condición (*while*) para buscar en todos los cursos que llevó el alumno y dos condiciones (*if*) para identificar si fue o no calificado. La complejidad ciclomática de este código es por lo tanto de 4. Los casos de prueba para este código serían:

1. No hay cursos llevados por el estudiante (la condición del while es falsa)
2. Hay un solo curso y no coincide con c1 (la condición del while es verdadera y del primer if falsa)
3. Hay un curso, coincide con c1 y ya está calificado (las condiciones del while y los dos if son verdaderas)
4. Hay un solo curso, coincide con c1 y no está calificado (las condiciones del while y del primer if son verdaderas y la del segundo if es falsa)

```
//Revisar si un estudiante ya tiene calificado un curso con anterioridad
Public boolean yaCalificado (Curso c1, cursos) {
1       Boolean calificado = falso;
        // cursos es la lista de los  cursos llevados por el estudiante
2       while (cursos del estudiante por revisar) {
        // se revisan todos los cursos llevados por el estudiante
            cursos c2 = otro curso llevado
3           if curso1 = curso2 {
4                   if (c2.getCalificacion no es " null"){
5                           calificado = true;
6                           break;
7                   }               // fin del segundo if
8           }                       // fin del primer  if
9       }                           // fin del while
10  return calificado;
11  }
```

Figura 8. 4. Seudo-código de un método a ser probado.

El plan de prueba del método de la figura 8. 4 es:

#	Casos de prueba	Resultado esperado
1	No hay cursos llevados por el estudiante (la condición del while es falsa)	falso
2	Hay un solo curso y no coincide con c1 (la condición del while es verdadera y del primer if falsa)	falso
3	Hay un solo curso, coincide con c1 y ya está calificado (las condiciones del while y los dos if son verdaderas)	verdadero
4	Hay un curso, coincide con c1 y no está calificado (las condiciones del while y del primer if son verdaderas y la del segundo if es falsa)	falso

Pruebas caja negra

Este tipo de prueba, también conocida como *prueba funcional*, revisa que la unidad cumpla con la funcionalidad esperada sin considerar el detalle del código. Para definir los casos de prueba, se establecen los posibles resultados esperados de la unidad y se identifican los conjuntos de valores de los parámetros, para que se generen estos resultados.

Una técnica para diseñar los casos de prueba de caja negra son las *Tablas de decisión*. Las tablas de decisión ayudan a describir combinaciones de datos de entrada que generan diferentes salidas.

Una tabla de decisión tiene 2 secciones:

Condiciones de parámetros de entrada: En la parte superior de la tabla se define la lista de condiciones de los parámetros de entrada y sus posibles combinaciones de valores cierto y falso.

Resultados esperados: Las dos secciones de abajo especifican los resultados esperados para las combinaciones de valores de las condiciones de entrada. Se marca con X que resultado se espera para cada posible combinación de valores de los parámetros.

En la figura 8. 5 se muestra la tabla de decisiones para el seudo-código de la figura 8. 4.

Condiciones de parámetros de entrada			
c1 es uno de los cursos del estudiante	verdadero	verdadero	falsa
c1 tiene calificación	verdadero	falsa	falsa
Resultados esperados			
El método yaCalificado sea verdadero	*X*		
El método yaCalificado sea falso		*X*	*X*

Figura 8. 5. Ejemplo de tabla de decisión.

El Plan de pruebas unitarias de caja negra, creado a partir de la tabla de decisión, es una tabla que tiene como casos de prueba la combinación de condiciones de valores cierto y falso para los parámetros de entrada y los resultados esperados. En la figura 8. 6 se muestra el Plan de pruebas para la tabla de decisiones.

#	Casos de prueba	Resultado esperado
1	C1 está en los cursos y c1 tiene calificación	verdadero
2	C1 está en los cursos y no tiene calificación	falso
3	C1 no está en los cursos y no tiene calificación	falso

Figura 8. 6. Plan de prueba de caja negra.

Las tablas de decisión es una técnica que se puede emplear durante el desarrollo del software, en diferentes etapas, cuando hay combinaciones de condiciones para ayudar en la toma de decisiones.

Efectuar las pruebas

Para efectuar las pruebas unitarias, se recomienda primero ejecutar los casos de prueba de caja negra. Posteriormente se agregan otros casos de prueba que permitan revisar la estructura, según la técnica de caja blanca.

Muchas veces al querer probar una operación o método requiere de otros que no están disponibles. Para simular su comportamiento se declaran métodos sustitutos llamado **Stubs**. Los sustitutos no hacen nada excepto regresar el valor esperado, por lo que se puede programar una pequeña interfaz por el cual el programador simula que se invocó el método y que eventualmente proporciona el resultado esperado.

Otro tipo de sustituto es un **Drive**r que es un programa que inicializa variables no locales y los parámetros para activar a la unidad que se está probando.

Los defectos encontrados en las pruebas se registran en la forma Registro de defectos y se corrigen. Se repite el caso de prueba correspondiente para asegurarse que el defecto quedó corregido.

Una vez efectuadas las pruebas unitarias, cada ingeniero de desarrollo reporta el tiempo que se tardó en efectuarlas en su forma Semana personal.

8.5 Profundización de temas para el segundo ciclo

8.5.1 Programación extrema XP

XP (www.programacionextrema.org)es un método que promete reducir riesgos del proyecto, mejorar respuesta a cambios en el negocio, mejorar la productividad y agregar satisfacción al equipo de desarrollo.

Los supuestos de XP son:

1. Interacción con el cliente
2. Planeación del proyecto
3. Iteraciones
4. Estrategia de diseño
5. Pruebas
6. Desarrollo
7. Reinicio del ciclo.

Los valores fundamentales son:

1. Comunicación: Es el primer valor y es el medio del aprendizaje y la trasmisión de ideas.
2. Simplicidad: consiste en mantener el proyecto en un estado que combine funcionalidad y la mayor sencillez posible.
3. Retroalimentación: evaluar constantemente el proyecto por todos los involucrados mediante la realización de pruebas, constante comunicación y ciclos cortos de desarrollo.
4. Corage o valentía: fomentar proyectos sencillos, sometidos a continuas evaluaciones y realizar modificaciones constantes.

Sus prácticas:

1. Juego de la planeación
2. Entregas pequeñas
3. Metáfora
4. Diseño simple

5. Pruebas
6. Reorganización, refactorización
7. Programación por pares
8. Propiedad colectiva
9. Integración continua
10. 40 horas semanales
11. Cliente en el sitio de desarrollo.
12. Estándares de codificación

En que se distingue de otros métodos:

- Busca una retroalimentación, temprana, concreta y constante a través de ciclos cortos.
- Tiene un enfoque de planeación incremental.
- Facilita la reprogramación incorporando funcionalidades según necesidades cambiantes del negocio.
- Se basa en la comunicación oral, pruebas y código fuente para informar sobre la estructura del sistema y su intención.
- Se basa en el proceso de diseño evolutivo que dura toda la vida del sistema.
- Se basa en una colaboración muy cercana de programadores con habilidades normales.
- Se basa en las prácticas que comparten el instinto a corto plazo de programadores con el interés a largo plazo del proyecto.
- Confianza en casos de pruebas desarrollados por los programadores y por los clientes para monitorear el progreso y encontrar defectos tempranamente.

8.5.2 Desarrollo basado en componentes

El desarrollo basado en *componentes* surge a finales de los 90´s como un "enfoque de reutilización para el desarrollo de software" [Sommerville p. 310]. Surge como ampliación a la expectativa de reutilización que generó la orientación a objetos. Los componentes son más generales que las clases u objetos, de hecho son conjuntos de ellos que proporcionan servicios. Los componentes pueden estar escritos en una variedad de lenguajes de programación y ser transparente a quien los invoca.

Para que el componente suministre un servicio se requieren de algunas características:

- Que el componente sea una unidad ejecutable independiente. No se compila como los otros componentes.
- Cada componente indica su interfaz y las interacciones son a través de ella. La interfaz son operaciones parametrizadas.
- Un componente puede tener una interfaz para suministrar (de salida) que define los servicios que proporciona y otra para solicitar (de entrada) algún servicio.

Existen componentes de diferentes niveles de abstracción, desde funciones simples como calcular una raíz cuadrada hasta aplicaciones completas.

El incorporar el desarrollo orientados a componente se inicia después de definir la arquitectura del sistema e incluye las siguientes actividades:

- Especificar los componentes.
- Buscar componentes.
- Incorporar los componentes en el diseño.

Muchas veces ya se tienen los componentes y entonces lo que se hace es diseñar en base a los componentes disponibles.

Las ventajas del desarrollo basado en componentes son:

- Disminución del tiempo de desarrollo.
- Aumento de la confiabilidad del sistema.
- Entrega más rápida del sistema.
- Costos más bajos al usar cosas ya hechas.

La desventaja es que el diseño puede ser menos eficiente que el hecho a la medida.

Las aplicaciones web están promoviendo este tipo de desarrollos pues existen componentes o servicios ya probados para muchas de las características típicas de estos sistemas.

Marcos de trabajo (Frameworks)

Los marcos de trabajo (framework) son subsistemas compuestos de una colección de clases abstractas y las interfaces entre ellas. [Sommerville p. 314].

Para construir una aplicación concreta lo que se hace es implementar las clases abstractas y agregar los componentes necesarios a la aplicación. Existen marcos de trabajo para aplicaciones web en Java que facilitan el desarrollo de estos tipos de aplicaciones que siguen el patrón MVC como es el J2EE que es un marco de trabajo para aplicaciones empresariales.

A menudo los marcos de trabajo incluyen de varios patrones de diseño.

El problema central de estos marcos de trabajo es su dificultad para entenderlos, pero facilitan y agilizan la construcción de las aplicaciones.

8.5.3 Patrones de diseño

Los patrones de diseño son otra forma de promover el reuso al favorecer incorporar soluciones probadas a problemas específicos. Su ámbito de trabajo es en el diseño de los paquetes. Para encontrar el patrón aplicable a un problema es útil tener una clasificación según los niveles de abstracción o los problemas que resuelven. Cada autor tiene diversas clasificaciones.

[Buschmann p. 222] propone las siguientes categorías:

- **Descomposición estructural**. Ayudan a descomponer paquetes o subsistemas en partes cooperativas. Ejemplo: *Todo-parte (Whole-part*).
- **Organizar el trabajo**. Definen partes que colaboran entre sí para resolver problemas complejos. Ejemplo: *Maestro-esclavo (Master-slave*) .
- **Controlar el acceso**. Proporcionan el acceso a servicios o componentes. Ejemplo: *Proxy*.
- **Administración**. Manejan colecciones de objetos, servicios y componentes homólogos. Ejemplo: *Procesador de comandos (Command Processor*).
- **Comunicación**. Organizan la comunicación entre componentes. Ejemplo: *Publicador-suscriptor (Publisher-suscriber*).

Otra clasificación es la dada por [Stelting]:

- **Patrones de creación**. Sirven para crear objetos en un sistema. Ejemplo: Fábrica abstracta (Abstract factory), Constructor (Builder), Prototipo (Prototype)
- **Patrones de comportamiento**. Se ocupan del control de flujo en un sistema. Ejemplos: Comando (Command), Iterador (Iterator), Observador (Observer).
- **Patrones estructurales**. Describen formas efectivas para partir o combinar elementos de una aplicación. Ejemplos: Puente (Bridge), Decorador (Decorador), Fachada (Facade), Proxy.
- **Patrones de sistemas**. Son patrones de arquitectura. Ejemplos: Modelo-Vista-Controlador (Model-View-Controller), Sesión (Session), Transacción (Transaction).

Estos patrones se pueden incluir al diseñar los paquetes con detalle incorporando soluciones probadas. Generalmente usar patrones hace los diseños menos eficientes, pero más adaptables a cambios en las necesidades del sistema.

8.5.4 Herramientas y entornos de programación

Para el desarrollo orientado a objetos existen herramientas que facilitan la generación del código. Por ejemplo, se puede usar un diagramador de UML que incluya la posibilidad de que a partir de los diagramas, se genere código para algunos lenguajes de programación. Estas herramientas lo que promueven es que el desarrollador pase más tiempo refinando y completando los diagramas que trabajando en el compilado de su código. Estas herramientas generan esqueletos de código con los que el programador debe generar el código completo.

Una ventaja extra de este tipo de herramientas es que los diagramadores pueden verificar la consistencia de los diferentes diagramas.

Otra característica de estas herramientas es que es posible generar los diagramas a partir del código, a esto se le llama hacer *Ingeniería inversa* y puede ser útil cuando no se siguió el proceso de desarrollo y se construyó el código, de esta manera se obtienen sus diagramas.

Ejemplos de estas herramientas son Racional Rose, Together, Visio, etc. Pero hay otras herramientas para agilizar la programación como contar con ambientes de desarrollo con bibliotecas de componentes para incluir en la aplicación. Principalmente se pueden usar ambientes para el desarrollo de la capa de interfaz de usuario en que se arrastra y pegan componentes como botones, listas desplegables, etc.

Lo mismo sucede para la capa de almacenamiento cuando se utiliza una base de datos relacional. Ejemplos son Oracle, Sybase, etc.

Referencias bibliográficas

- Beck K., *Extreme Programming Explained,* Addison Wesley, 2000.
- Braude E. J. *Ingeniería de software. Una perspectiva orientada a objetos*. Alfaomega 2003.
- Humphrey Watts S., *"Introduction to Team Software Process"*. SEI Series in Software Engineering, Addison Wesley, 2000.
- Pressman R. S. *Ingeniería de software. Un enfoque práctico*. 2ª edición 1987
- Sommerville I. *Ingeniería de Software.* 6ª edición. Addison wesley 2002.
- Stelting S., Maassen O. *Applied Java Patterns.*Sun Microsystems/Prentice Hall. 2002.

Capítulo 9

Fase de Integración y prueba del sistema

Objetivos del capítulo:

Describir las actividades necesarias para la fase de Integración y prueba del sistema.

9.1 Fase de Integración y prueba del sistema: objetivos, actividades y productos.

El objetivo de esta fase es preparar la integración del software, hacer la integración del sistema y probar. Se prueba que el sistema cumpla con sus requerimientos y se escriben los manuales.

Objetivo:

O1. Preparar la integración del sistema.
Se revisa el orden de integración de los componentes especificado en el Plan de pruebas de integración con la finalidad de reflejar algún cambio realizado durante la fase de Construcción y se verifica que los componentes cumplan con los requisitos necesarios para su integración.

Actividades:

A1.1 Asegurarse que se encuentren todos los componentes en la última versión probada.

A.1.2 Revisar Plan de pruebas de integración, específicamente el orden de integración de los componentes y realizar los cambios necesarios.

Producto:

- Plan de pruebas de integración revisado.

Objetivo:

O2. Realizar la integración del sistema y probarla.
Se realiza la integración del sistema siguiendo el Plan de pruebas de integración.

Actividad

A2.1 Integrar el sistema y probarlo.

Productos:

- Reporte de las pruebas en Registro de defectos

Objetivo:

O3. Realizar la prueba del sistema.
Revisar el Plan de prueba del sistema definido en la fase de Especificación de requerimientos y actualizarlo, en caso de que sea necesario. Efectuar las pruebas del sistema siguiendo el Plan.

Actividad

A3.1 Probar el sistema siguiendo el Plan.

Productos:

- Informe de la prueba del sistema.

Objetivo:

O4. Construir los manuales del sistema.
Para completar la documentación del sistema se hacen los manuales necesarios como: el de usuario, de instalación, etc.

Actividad

A4.1 Hacer los manuales necesarios del sistema.

Productos:

- Manuales de usuario.
- Manual de operación o instalación.
- Manual de desarrollo y mantenimiento.

Formas a entregar en todas las fases:

- Semana personal
- Semana del equipo
- Minuta de la reunión
- Lista de verificación
- Registro de defectos

Resumen de productos a entregar:

Producto	Responsable
Reporte de las pruebas en la forma de Registro de defectos	Administrador de desarrollo y de apoyo
Informe de la prueba del sistema	Administrador de desarrollo e Ingeniero de desarrollo.
Manuales de usuario Manual de operación o instalación Manual de desarrollo y mantenimiento	Administrador de desarrollo e Ingeniero de desarrollo

9.2 Preparar la integración del sistema

El objetivo de la integración es comprobar que se tienen todos los componentes y que funcionan bien juntos.

Cada ingeniero de desarrollo es responsable de la calidad de los componentes que genera, asegurándose que ha sido probado según lo planeado y que cumple con los estándares acordados.

Para hacer la integración, se usa como guía el diseño arquitectónico, y se asegura que se tienen todos los componentes definidos en el diseño en la última versión aprobada.

El administrador de apoyo deberá revisar estos componentes y asegurarse que se encuentren en la versión adecuada y proporcionarlos al equipo para la integración. Además será el encargado de identificar cuáles componentes podrán ser reutilizables y proponer los cambios necesarios (generalizaciones, especializaciones) para guardarlos para futuros desarrollos.

9.3 Integración del sistema

Para realizar la integración se parte de lo que se describe en Plan de pruebas de integración, el administrador de apoyo se asegura que las últimas versiones de los componentes se encuentren disponibles en el depósito del proyecto, el administrador de desarrollo se encarga de coordinar las actividades y todo el equipo participa. A esto se le conoce como *construir el sistema.* Conforme *se van integrando componentes se va probando que funcionen juntos adecuadamente.* Se proponen las pruebas que se aplicarán para verificar la correcta interacción entre estos componentes.

El elemento de un paquete que no dependa de otros, es el primero que se integra. En el caso de las clases, una clase depende de otra si invoca alguno de sus métodos. Bajo esta regla, se puede crear una gráfica de dependencias entre clases que sirve de guía para definir el orden de integración.

Al integrar los componentes se identifican los métodos que se invocan entre clases. Estos métodos deben ejecutarse para comprobar que el paso de parámetros y los resultados devueltos son los adecuados.

Estrategias para la integración del sistema

Para definir el orden de integración de los elementos del sistema, se puede seguir alguna de estas estrategias [Binder]

- **Estrategia de paquetes.** La idea es hacer las integraciones según los paquetes del diseño. Por ejemplo, se integran todos los elementos del paquete de interfaz de usuario para un actor.

  Este enfoque tiene la ventaja, de que se integran los componentes de un paquete y posteriormente se unen a otros paquetes ya integrados, siguiendo la arquitectura.

- **Estrategia por caso de uso.** La idea es integrar los paquetes de las tres capas que corresponden a la realización de un caso de uso. Se puede hacerlo para varios casos de uso en paralelo, cuando no hay dependencia entre ellos.

No existe una estrategia adecuada para todos los sistemas, decidir cuál usar es decisión del proyecto y el nivel de diseño efectuado.

Es importante llevar un registro de los defectos encontrados para facilitar su seguimiento, en la forma de Registro de defectos en que se anotan las observaciones sobre los mensajes mostrados o las pantallas resultantes. Se anota también, el responsable de arreglar cada defecto.

9.4 Prueba del sistema

En la prueba del sistema, se tratará de comprobar que el sistema cumple con todos los requerimientos establecidos tanto los funcionales como los no funcionales. El encargado de coordinar la ejecución del Plan de pruebas del sistema es el administrador de desarrollo con la participación de todo el equipo.

Como es muy difícil probar al cien por ciento todos los aspectos del sistema, se debe planear a qué se le dará prioridad. Por ejemplo, se puede decidir primero asegurar que cumple con sus funciones, luego evaluar la usabilidad, posteriormente comprobar el sistema bajo condiciones de estrés y medir el desempeño.

Se prueban los siguientes aspectos definiendo casos de prueba para:

- La instalación del sistema.
- El arranque del sistema.
- La usabilidad del sistema con la participación de usuarios reales.
- Se deben incluir pruebas de los otros requerimientos no funcionales como:
  - Rendimiento
  - Robustez – recuperación de errores de hardware
  - Tiempo de respuesta
  - Capacidad de carga (memoria, accesos)
  - Estimación de número de defectos por día
  - Estimación de tiempo de corrección de defectos
  - Estimación de tiempo total de prueba de sistema y corrección.

En ciclos posteriores se deberán hacer pruebas de regresión (ver 9.6.2) al incorporar nuevas funcionalidades para asegurar que funcionen bien con las anteriores.

Planeación de las pruebas

Para el plan de pruebas del sistema se toma como base los planes de prueba que se hicieron en las fases de requerimientos y de diseño. El plan de pruebas que debe incluir:

- Qué se va a probar.
- En qué orden.
- El material de prueba, esto es, el componente a probar, los datos de prueba, el software que se necesita.
- El resultado esperado para cada dato de prueba.
- Responsable de la prueba, los participantes en la prueba, la fecha y lugar en que se hará la prueba.
- Plantilla para el informe de los resultados obtenidos indicando el número de defectos encontrados.

Efectuadas las pruebas, se corrigen los defectos encontrados. Se vuelven a aplicar pruebas, conocidas como *pruebas de regresión*. Las pruebas de regresión sirven para detectar defectos al hacer cambios a componentes ya probados. Al iniciar un nuevo ciclo de desarrollo, se generan nuevos componentes o se modifican los que se tenían para incluir nuevas funcionalidades o características. Esos componentes pueden ocasionar fallas en lo que ya funcionaba por efectos laterales o nuevas interacciones.

Si el sistema pasa todas las pruebas, se libera y entrega al cliente.

9.5 Construcción de manuales

Al construir un sistema, siguiendo este ciclo de desarrollo, se ha generando la documentación del sistema para los desarrolladores en siguientes ciclos o para el mantenimiento. Sin embargo, generalmente se requiere de manuales para otros involucrados en el sistema, principalmente sus usuarios.

Los manuales se deben organizar según las necesidades de la persona que lo va a leer, no de la estructura del sistema. Deberán estar escritos en el lenguaje del lector a quien está dirigido.

Cada manual deberá incluir:

- Un índice de temas
- Un glosario de términos
- Tabla de contenido detallado
- Secciones de mensajes de error, problemas y cómo resolverlos.

El coordinador de la creación de manuales es el administrador de desarrollo con la participación de todo el equipo.

Hay distintos tipos de lectores de documentación:

- el cliente que lo encargó
- el usuario final
- quien instalará el sistema
- quien lo mantendrá
- quien lo desarrolló
- quien administra el proceso de desarrollo.

Manual de usuario

Este manual debe contener información para usar el sistema. Es un documento electrónico o impreso que describe la forma de uso del software con base a la interfaz de usuario [Moprosoft]. Debe estar escrito en un lenguaje que pueda entender el usuario. El contenido debe incluir: cómo se accede al sistema, cuáles son las interfaces de usuario que presenta el sistema, qué se espera en cada opción o campo, cuáles son los mensajes de respuesta el sistema y cómo resolver posibles problemas en la ejecución. También, es posible que se requiera incluir información sobre la instalación, en este manual, que pueda ser útil al usuario.

Es importante aplicar pruebas de usuario al manual para asegurarse que la escritura sea clara, precisa, completa y entendible.

Manual de operación o instalación

Debe emplearse un lenguaje que pueda ser entendible por el encargado de la instalación del software, por lo que es importante especificar los pre-requisitos de instalación. Puede estar en formato impreso o electrónico [Moprosoft]. Su contenido será:

- La versión que se está instalando.
- Necesidades de hardware indicando capacidades y velocidades mínimas.
- Necesidades de software con versiones mínimas.
- Procesos para la instalación.
- A quién recurrir en caso de problemas.

Manual de desarrollo y mantenimiento

Este manual contiene el objetivo y características del software, las decisiones tomadas en el desarrollo y la historia de toda la documentación generada. Incluye los documentos de las fases de captura de requerimientos, diseño, implementación y pruebas. Deberá estar escrito en términos comprensibles para el personal de mantenimiento [Moprosoft].

Si se siguió el proceso adecuadamente, ya se ha generando la documentación en la creación del sistema, y estará escrito en términos técnicos para uso de los ingenieros de desarrollo y mantenimiento.

Manual de administración del proceso de desarrollo

El propósito es permitir enterarse en forma somera de la administración del proyecto de desarrollo de este sistema. Aparece el nombre del sistema, el cual debe ser único en la organización y deberá conservarse invariable en todos los documentos referentes a ese sistema.

Contiene la información sobre la administración del proceso que comprende las fases de lanzamiento, estrategia, planeación y el cierre de cada ciclo.

La información necesaria en este manual es:

- Contenido.
- Nombre del sistema.
- Equipo encargado del sistema.
- Nombre del personal participante en el desarrollo del sistema.
- Resumen Administrativo.

Revisión de los manuales

Se aconseja que uno o más colegas revisen los manuales generados. A continuación se presentan elementos para las revisiones.

- *Organización de los manuales*.

¿El manual está organizado de acuerdo a lo que el lector necesita o de acuerdo al contenido del producto?

- *Terminología del manual*.

¿El manual presume conocimiento que el lector probablemente no tenga? Definir cualquier palabra no clara en el glosario.

- *Contenido del manual.*

¿El manual cubre el contenido para las necesidades requeridas?

- *Precisión.*

¿El desarrollo de los métodos y procedimientos es de acuerdo a su descripción?

- *Legibilidad.*

¿Es fácil leer el manual? Se verifica que es fácil entender lo que está escrito.

- *Entendimiento.*

¿Las personas entienden lo que está escrito? Generalmente, esta pregunta es la más difícil de responder.

Una buena prueba a los manuales, consisten en solicitar a alguien que no ha utilizando el sistema que lo haga siguiendo el manual. Se observan sus reacciones, registran los problemas y modifica el manual de acuerdo a los problemas identificados.

9.6 Profundización de temas para el segundo ciclo

9.6.1 Pruebas ágiles

Una de las prácticas claves de XP y otros procesos ágiles es *el desarrollo dirigido por pruebas (Test-driven development)* que consiste en escribir las pruebas antes del código [Larman p. 292].

Escribir pruebas es algo aburrido y tedioso, sin embargo, escribir primero las pruebas y posteriormente el código que deberá pasar esa prueba puede ser más creativo. Por ejemplo, si se va a construir una clase **A** con una operación *z*, se puede primero escribir una clase que pruebe **A** que tenga dos operaciones, la que pruebe la operación *z* y otra que compare el resultado esperado de *z* con el obtenido realmente. Este enfoque tiene la ventaja de que automatiza la prueba y no requiere intervención humana.

Otra ventaja de escribir el código para pasar una prueba, es que se genera un pensamiento de *prueba de caja negra* que le ayuda en el diseño de la clase.

Estas pruebas serán parte del proceso de prueba unitarias y de integración que se pueden aplicar al hacer nuevas versiones del código.

Existen algunas herramientas y marcos de trabajo de código abierto para apoyar este tipo de pruebas.

9.6.2 Pruebas de regresión.

Las pruebas de regresión deben hacerse en procesos iterativos después de las pruebas unitarias, como primer paso en la integración y en el mantenimiento de software. [Binder p. 756-770]

Las pruebas de regresión sirven para detectar defectos al hacer cambios a componentes ya probados pero que se incorporan a ambientes con cambios. Esta situación se presenta cuando se tienen iteraciones de desarrollo o al hacer el mantenimiento al software.

Al iniciar una nueva iteración, se generan nuevos componentes o se modifican los que se tenían para incluir nuevas funcionalidades o características. Esos componentes pueden ocasionar fallas en lo que ya funcionaba por efectos laterales o nuevas interacciones. Un componente en línea base ya ha pasado un conjunto de pruebas. Una versión *delta* de un componente, es una modificación a ese componente que no ha pasado una prueba de

regresión. Una *construcción delta* es una configuración ejecutable que contiene componentes en línea base y deltas que deberán funcionar adecuadamente.

Mantenimiento de software

Hay 4 tipos de mantenimiento al software: *correctivo* se refiere a los cambios que se hacen al detectar un defecto en un software, generalmente se refiere a cuando ya está en uso. *Adaptativo* son los cambios que se hacen cuando se ajusta a nuevos ambientes o a otros sistemas. *Perfectivo* son cambios que se requieren para mejorar sus capacidades. *Preventivo* son cambios para aumentar sus características.

En todos estos casos se hacen pruebas de regresión para detectar defectos causados por los cambios como efectos laterales o por correcciones mal hechas.

El proceso típico de pruebas de regresión consiste en:

- Definir nuevos casos de prueba que revisen las interacciones en la nueva versión tanto para los componentes modificados como para los efectos laterales de esas modificaciones.
- Efectuar las pruebas.
- Tomar las acciones necesarias cuando no se pase una prueba.

Las pruebas de regresión deberán hacerse siempre se tenga uno de estos casos:

- Se desarrolle una nueva subclase, se deberá probar la superclase también.
- Se cambie una superclase, se probarán sus subclases también.
- Al cambiar una clase servidora, se probarán sus clientes.
- Al detectar algún defecto.
- Al crear una nueva versión de un software. Se deberá probar cada nueva característica.
- Se haga un nuevo incremento al software.
- Se tenga una versión estable y se libere el software.

Referencias bibliográficas

- Binder R. V. *"Testing Object-oriented Systems. Models, patterns and Tools"*. Addison Wesley 2000.
- Braude E. J. *Ingeniería de software. Una perspectiva orientada a objetos*. Alfaomega 2003. p 432-440
- Humphrey Watts S., "*Introduction to Team Software Process*", SEI Series in Software Engineering, Addison Wesley, 2000.
- Larman C. "*Agile & Iterative Development. A Manager's guide*". Addison Wesley 2004.
- Moprosoft Modelo de Procesos para la Industria de Software. Versión 1.1 Mayo 2003.
- Sommerville I. *Ingeniería de Software.* 6ª edición. Addison Wesley 2002.

Capítulo 10

Fase de Cierre

Objetivos del capítulo:
Describir las actividades necesarias para la fase de cierre y la entrega del sistema.

10.1 Fase de Cierre: objetivos, actividades y productos.

El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso y del estado de la configuración.

Objetivo:

O1. Evaluar al personal y al equipo en el ciclo de desarrollo.
Terminado un ciclo de desarrollo se hace una evaluación personal del desempeño de cada miembro del equipo y una evaluación entre colegas del desempeño del equipo.

Actividad

A1.1 Evaluar el cumplimiento de los objetivos personales.
A1.2 Evaluar el desempeño del equipo y la asignación de roles.

Productos:

- **Forma de evaluación del equipo y personal**

Objetivo:

O2. Registrar las lecciones aprendidas y hacer propuestas de mejora.
Se registran las lecciones aprendidas y se buscan posibles mejoras para el siguiente ciclo.

Actividades:

A2.1 Registrar las mejores prácticas, problemas recurrentes y experiencias exitosas en la forma de Lecciones aprendidas.
A2.2 Hacer sugerencias de lo que podría mejorarse para el siguiente ciclo.

Productos:

- **Lecciones aprendidas y sugerencias de mejoras.**

Objetivo:

O3. Hacer los informes de mediciones del proceso y del estado de la configuración.
Se hace un informe de las mediciones del proceso seguido y del estado de la configuración para que sirva de base en ciclos posteriores de desarrollo.

Actividades:

A3.1 Hacer el Informe de mediciones
A3.2 Hacer el Informe del estado de la configuración.

Productos:

- **Informe de mediciones.**
- **Forma de Informe del estado de la configuración**

Formas a entregar en todas las fases:

- Semana personal
- Semana del equipo
- Minuta de la reunión
- Lista de verificación
- Registro de defectos

Resumen de productos a entregar:

Producto	Responsable
Forma de evaluación del equipo y personal	Líder del equipo
Lecciones aprendidas y propuestas de mejora.	Líder del equipo y Administrador de desarrollo
Informe de mediciones	Administrador de calidad y planeación
Informe del estado de la configuración	Administrador de apoyo

10.2 Evaluación personal y del equipo

Cada miembro del equipo revisa sus objetivos personales definidos y documentados en la fase de Lanzamiento y comprueba si se alcanzaron. Para los que no se alcanzaron se busca la razón y le sirve de reflexión para el siguiente ciclo de desarrollo.

La evaluación del equipo se hace a través de la forma de *Evaluación del equipo y personal* (Ver Forma). Cada miembro del equipo llena una copia de esta forma y evalúa en una escala del 1 (bajo) al 5 (alto) los criterios propuestos para cada rubro. El líder del equipo pide a los miembros del equipo que sean honestos e imparciales en sus evaluaciones, pues a los que podrán ayudar o perjudicar, es a ellos mismos como equipo.

Esta forma tiene un encabezado en el que se pone el nombre del equipo y el ciclo.

La forma consta de una tabla para evaluar al equipo según los criterios de: espíritu de equipo, efectividad del equipo, experiencia adquirida, productividad del equipo, calidad del proceso y calidad del producto.

Evaluación del equipo sobre los siguientes criterios. Encierre un número desde el 1 (bajo) hasta el 5 (alto)					
Espíritu de equipo	1	2	3	4	5
Efectividad del equipo	1	2	3	4	5
Experiencia adquirida	1	2	3	4	5
Productividad del equipo	1	2	3	4	5
Calidad del proceso	1	2	3	4	5
Calidad del producto	1	2	3	4	5

Posteriormente se evalúan los roles en tablas semejantes. Cada miembro evalúa qué tanto (porcentaje) esfuerzo se requiere de cada rol y qué tan difícil cree que fue. La contribución de cada rol se evalúa en la segunda tabla. En la tercera tabla se evalúa el apoyo o ayuda que proporcionó cada rol en el proceso. En la última tabla se evalúa el desempeño de cada persona en el rol.

Como cada rol evalúa a todos los roles, en particular el propio, hay una autoevaluación. Esta concepción del desempeño personal se compara con la visión del resto de los miembros y permite la autorreflexión en el desempeño. Estas evaluaciones se promedian y se puede obtener una perspectiva del desempeño del equipo.

Otro resultado importante que se obtiene de esta evaluación es una reflexión y conocimiento de parte de cada miembro del equipo que podrá ayudarle en su vida personal y profesional.

10.3 Lecciones aprendidas y sugerencias de mejoras

En esta última fase se revisa el trabajo realizado y se toma conciencia de lo aprendido, se reflexiona en qué se podría mejorar para el siguiente ciclo, documentándolo en la forma de *Lecciones aprendidas y sugerencias de mejoras* [MoProSoft p. 97].

En la forma de *Lecciones aprendidas y sugerencias de mejoras*, el equipo documenta lo que les funcionó mejor en el proceso de desarrollo, las mejores prácticas, las experiencias exitosas y los problemas recurrentes a los que se enfrentaron. El líder del equipo y el

administrador de desarrollo son los responsables de guiar al equipo en el llenado de esta forma.

La forma propuesta tiene un encabezado como las otras formas que debe llenar el equipo. Consta de 3 tablas para documentar las lecciones aprendidas con respecto a las mejores prácticas, los problemas recurrentes y las experiencias exitosas. Esto significa que esta forma contiene tanto buenas prácticas que se aprendieron como cuestiones que no deberán repetirse.

Mejores prácticas

Del equipo	Del proceso

Problemas recurrentes

Del equipo	Del proceso

Experiencias exitosas

Del equipo	Del proceso

El líder del equipo solicita a cada rol que identifique lecciones que se aprendieron de acuerdo a sus responsabilidades para los tres rubros se registran en la columna del equipo. El administrador de desarrollo registra las lecciones aprendidas en la columna del proceso. Posteriormente hacen la síntesis y la documenta en la forma en alguna de las tablas o como comentarios.

- **Líder de equipo** analizará el desempeño del equipo desde la perspectiva de liderazgo. Se documenta en las columnas del equipo de la forma de Lecciones aprendidas.
- **Administrador de desarrollo** debe comparar el contenido del producto con los requerimientos y evaluar la efectividad de la estrategia de desarrollo. Se documenta en las columnas del proceso de la forma de Lecciones aprendidas
- **Administrador de planeación** hará una comparación entre el desarrollo real del equipo y lo que se planeó. Se puede documentar las experiencias tanto en la columna del equipo como del proceso.
- **Administrador de calidad** comparará los datos actuales de calidad para describir el desempeño del equipo. Se puede documentar las experiencias tanto en la columna del equipo como del proceso.
- **Administrador de apoyo** evaluará las herramientas utilizadas, el uso de repositorio compartido y el control de versiones. Se puede documentar las experiencias tanto en la columna del equipo como del proceso.

La forma incluye un espacio para poner otros comentarios que sean relevantes sobre lo aprendido en ese ciclo de desarrollo.

El objetivo de este documento es dejar constancia de lo aprendido e ir conformando la *Base de conocimiento* del equipo para proyectos futuros.

Sugerencias de mejoras

Durante la reflexión sobre el desarrollo del ciclo surgen las propuestas de cómo mejorar el desempeño del equipo. Estas propuestas pueden referirse tanto al proceso como al equipo. Cada propuesta sugerida por algún miembro del equipo es analizada por todos y de ser aprobada se registra en la forma correspondiente.

Al hacer las propuestas de mejoras al proceso, se sugiere que no se promuevan grandes cambios de un ciclo a otro, más bien pequeñas mejoras que **verdaderamente** se pueden llevar a cabo. Con pequeños cambios el equipo puedo observar cómo se facilita el trabajo lo que significa que se incrementa el nivel de madurez del equipo de desarrollo.

Como parte de las mejoras al equipo, se reflexiona sobre mantener o cambiar los roles para el siguiente ciclo. Ambas posturas tienen ventajas. Si los roles se mantienen, seguramente se mejorará su desempeño pues cada miembro ya conoce sus responsabilidades y tiene experiencia en su ejecución. Si se cambian, cada quien aprenderá otro rol.

Para las mejoras de proceso se pueden incorporar prácticas de modelos de de procesos de referencia como MoProSoft o CMMI, que se verán en la sección 10.5.1. Siguiendo las prácticas propuestas en este curso se alcanza aproximadamente el nivel 2 de madurez de estos modelos.

10.4 Informes del sistema

Uno de los propósitos fundamentales del proceso es hacer mediciones constantes a los tiempos en efectuar las actividades, los tamaños de los productos y la búsqueda de defectos. En la fase del cierre se hace el resumen de todos estos datos para aprovechar esta experiencia en los ciclos siguientes.

Informe de mediciones

La forma de *Informe de mediciones* tiene por objetivo hacer la síntesis de varias mediciones que se hicieron durante todas las fases: los tiempos en efectuar cada fase, el tiempo total en horas del ciclo, los tamaños de cada producto, la cantidad de defectos que se encontraron en los productos, el total de defectos que se encontraron, la productividad y calidad de los productos.

Los responsables del llenado son los administradores de calidad y de planeación. La forma consta de 4 tablas que recolectan y resumen los datos de las formas que se llenaron en todo el ciclo:

- **El tiempo que se llevó cada fase en realizarse**. Este dato lo puede obtener el administrador de planeación de las formas semanas del equipo para sacar el número de horas/hombre de cada fase y el total del ciclo.

Fases	**Resumen de tiempos**
Lanzamiento	
Estrategia	
Planeación	
Especificación de Requerimientos	
Diseño	
Construcción	
Integración y prueba del sistema	
Tiempo total del ciclo	

- **El tamaño de los productos**. Estos datos están en la formas Semana del equipo.

Tamaños de los productos

Productos	**Tamaños**	**Unidades**

- **El número de defectos encontrados por producto**. Esta información la tiene el administrador de calidad de las reuniones de revisiones entre colegas y las formas de Registro de defectos y las correcciones realizadas por el instructor.

Defectos encontrados por producto

Producto	**Número de Defectos**

- **La productividad y calidad de los productos**. La productividad se obtiene de dividir el tamaño del producto entre el tiempo de desarrollo del producto. La calidad de los productos es la razón entre el número de defectos y el tamaño del producto.

Productividad y calidad de los productos

Producto	**Tamaño del producto / tiempo total de desarrollo**	**Número de defectos / Tamaño de producto**

Informe del estado de la configuración.

Antes de instalar el sistema en el lugar del cliente, es importante hacer una recopilación de las versiones adecuadas de los componentes del sistema. Es por esto que es importante

llenar la forma del *Informe del estado de la configuración*. (Ver forma). El responsable de hacer este informe es el administrador de apoyo y el objetivo es revisar que se integre el producto final con los componentes en versiones adecuadas.

Esta forma consta de una tabla en que se listan los elementos de la configuración del sistema con los que se instalará. Se indica la versión y la dirección de donde deberán tomarse. Al hacer esta lista se está haciendo una auditoria lógica de los elementos que se deberán instalar y que sean las versiones correctas. En ciclos posteriores se podrá revisar a partir de qué versiones se hacen las modificaciones.

Elementos de la configuración al final del ciclo:

Elemento de la configuración	Versión	Dirección donde se encuentra

10.5 Profundización de temas para el segundo ciclo

10.5.1 Modelos de calidad

El desarrollo de software se ha vuelto una importante oportunidad para las naciones en cuanto a industria no contaminante y fácil de instalar (solo requiere personas pensando y trabajando en una computadora). Esto ha ocasionado la necesidad de evaluar la capacidad de cada organización que desarrolla software. Esta capacidad se mide principalmente en dos aspectos: la calidad del producto que genera y el proceso que sigue para desarrollarlo.

Varias organizaciones mundiales como la ISO o locales como el Software Engineering Institute de EU con el CMMI, o la Secretaría de Economía en México con Moprosoft, se han propuesto desarrollar modelos de procesos que permitan evaluar la madurez de las empresas que desarrollan software.

Estos modelos tienen dos aplicaciones importantes: ayudar a las organizaciones a medir su madurez y servir de marcos de referencia para incrementar esa madurez.

En México los modelos más populares son el ISO 15504 que es una norma mundial específica para software; el CMM o CMMI que son modelos de los Estados Unidos, nuestros socios comerciales principales. Moprosoft entró en vigencia como modelo y norma mexicana a partir del 2005.

Para saber más de estos modelos se pueden consultar:

Moprosoft: software.net.mx

10.5.2 Código de ética

La Ingeniería de Software es una profesión que comprende no solo habilidades técnicas sino responsabilidades sociales y legales. Los ingenieros de software como cualquier profesional deben comportarse de manera ética y moral responsables si quieren ser respetados como profesionales.

Las instituciones profesionales a nivel mundial como son la ACM y la IEEE, en la fuerza de tarea conjunta para la práctica profesional de la Ingeniería de Software, proponen un **Código de ética** que todo profesional de esta área deberá conocer. Para mayor de talle consultar: www.acm.org/serving/se/code.htm. En español se puede bajar de http://seeri.etsu.edu/SpanishVersionSECode.htm

Esté código tiene una versión corta que resume las aspiraciones de un profesional que todo profesional deberá consultar, leer y adaptar a sí mismos. En la versión completa se incluyen ejemplos de qué significan estas expresiones.

Los ingenieros de software deberán comprometerse ante sí mismos a esos principios que en forma resumida son:

1. *Público.* El ingeniero de software debe actuar de manera consistente con el interés público.
2. *Cliente y Empleador.* El ingeniero de software debe actuar cuidando de la mejor manera el interés del cliente y de su empleador y ser consistente con el interés público.
3. *Producto.* El ingeniero de software debe asegurar que su producto y sus posibles modificaciones cumplen con los mejores estándares profesionales posibles.
4. *Juicio.* El ingeniero de software debe mantener integridad e independencia en juicios profesionales.
5. *Administración.* Los ingenieros de software líderes y administradores deben aprobar y promover el comportamiento ético en la administración del desarrollo de software y su mantenimiento.
6. *Profesión.* El ingeniero de software debe aumentar la integridad y la reputación de la profesión de manera consistente con el interés público.
7. *Colegas.* El ingeniero de software debe ser leal y apoyar a sus colegas.
8. *Personal.* El ingeniero de software debe capacitarse continuamente durante toda la vida en las prácticas de su profesión y promover el enfoque ético al ejercer las prácticas de su profesión.

Referencias bibliográficas

- Braude E. J. *Ingeniería de software. Una perspectiva orientada a objetos*. Alfaomega 2003. p 432-440
- Humphrey Watts S., "*Introduction to Team Software Process*", SEI Series in Software Engineering, Addison Wesley, 2000.
- Moprosoft. *Modelo de Procesos para la Industria de Software*. Versión 1.1 Mayo 2003.

- "*Software Engineering Code of Ethics and Professional Practice*", IEEE-CS/ACM Joint Task on Software Engineering Ethics and Professional Practice, 1998. www.acm.org/serving/se/code.htm.
- En español de http://seeri.etsu.edu/SpanishVersionSECode.htm

Capítulo 11

Líder del equipo

11.1 Objetivos, habilidades y responsabilidades

Objetivos

- Construir y mantener un equipo efectivo.
- Motivar a todos los miembros a participar activamente y con entusiasmo en el proyecto.
- Resolver los conflictos que se presenten en el equipo. Si no es posible que llevan al instructor para que los resuelva.
- Mantener al instructor informado del progreso del equipo.
- Convocar y dirigir las reuniones del equipo de forma eficaz.

Habilidades

- Experiencia de liderazgo en grupos juveniles.
- Es un líder natural en los grupos.
- Conciliador de los intereses del grupo.
- Reconocimiento del grupo por su liderazgo.
- No conflictivo, motivador a cumplir los compromisos.
- Sociable y amistoso.

Responsabilidades

- Construye y mantiene el equipo y su efectividad.
- Motiva a los miembros a trabajar en equipo y cumplir sus compromisos.
- Resuelve los problemas del equipo y de las personas que lo integran.
- Informa al instructor sobre el progreso del equipo.
- Convoca y dirige las reuniones del equipo, prepara la agenda.
- Participa también en el equipo como Ingeniero de desarrollo.

11.2 Actividades del rol a realizar todas las semanas

- **Convocar al equipo a la reunión del equipo y preparar la agenda.**

El líder del equipo convoca a la reunión semanal y manda la agenda de los temas que se tratarán. Durante la reunión toma la minuta con los acuerdos.

Estos documentos deberán seguir el estándar de documentación e incluir: el nombre del equipo, fecha, hora de inicio, hora de terminación, lista de asistentes, asuntos a tratar, acuerdos y compromisos.

Se deberá generar, guardar y entregar al instructor, la agenda y la minuta de las reuniones semanales.

Producto:

Agenda y minuta de la reunión de esta semana.

- Asistir a la reunión con el instructor para informar del avance del equipo y entregarle los documentos.
- Recoger los documentos de la fase anterior revisados por el instructor, asignar a quién hará la corrección (si es el caso), entregarlos al administrador de apoyo para que los incorpore en la carpeta.
- Llenar la forma Semana personal.

11.3 Actividades específicas del rol en cada fase

- **Definición del equipo**

El líder del equipo es el único rol que entra en funciones en la primera semana, debido a que debe convocar al equipo a la primera reunión para integrar el equipo.

Actividades

- Convocar al equipo a la primera reunión del equipo y preparar la agenda.

Agenda y minuta de la primera reunión del equipo.

En esta reunión se decide:

Nombre del equipo
Logo
Lema
Día y lugar de reunión
Asignación de roles
Asuntos generales

Toma nota de los acuerdos y envía la minuta a todos los miembros y una copia se entrega al instructor.

Fase de Lanzamiento

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Actividades

- Dirigir al equipo en la definición de los objetivos del equipo y del producto.
- Conjuntar la definición de los objetivos de cada miembro del equipo.

- Definir los objetivos de su rol.
- Participar en la definición de los estándares.
- Participar en la identificación de riesgos.

Producto del que es responsable
- Documento con la definición de objetivos del equipo, producto, personales y de rol.

Agenda y minuta de la reunión semanal

La agenda que deberá tener como puntos importantes:
- Discutir y llegar al consenso de los objetivos del equipo y del producto.
- Invitar a los miembros a que definan sus objetivos personales y de sus roles. Ver capítulos de 11 al 16.
- Discutir la definición de los estándares de documentación.
- Discutir e identificar los riesgos que podría enfrentar el equipo para el proyecto.
- Asuntos generales.

Fase de Estrategia

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo ciclo en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Agenda y minuta de la reunión semanal

La agenda de la reunión en esta fase deberá tener como puntos importantes:
- Discutir la estrategia del equipo.
- Asuntos generales.

Fase de Planeación

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta misma fase se hace el introduce el concepto de calidad y se introducen las *revisiones entre colegas* como técnica de calidad que se harán a lo largo del desarrollo.

Agenda y minuta de la reunión semanal

La agenda deberá tener como puntos importantes:
- Discutir el Plan del equipo.
- Definir el plan de revisiones entre colegas.
- Asuntos generales

Fase de Especificación de requerimientos

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado que se establecen en el Plan de pruebas del software.

Agenda y minuta de la reunión semanal

La agenda que deberá tener como puntos importantes:
- Hacer la definición del problema y el glosario de términos.
- Discutir la Especificación de requerimientos.
- Discutir el Plan de pruebas del software.
- Asuntos generales.

Fase de Diseño

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración y establecer las clases con que se construirá el software en sus vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

Agenda y minuta de la reunión semanal

La agenda que deberá tener como puntos importantes:
- Discutir la especificación del ambiente de implementación.
- Discutir el estándar de documentación del diseño.
- Discutir la arquitectura con diagrama de paquetes y el diagrama de distribución.
- Discutir los diagramas de clases, de secuencia y de estados.
- Discutir el Plan de prueba de integración.
- Asuntos generales.

Fase de Construcción

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

Agenda y minuta de la reunión semanal

La agenda que deberá tener como puntos importantes:
- Discutir el diagrama detallado de las clases.

- Discutir la construcción del código, los estándares de codificación.
- Discutir las pruebas unitarias que se elaborarán.
- Asuntos generales.

Fase de Prueba del sistema

- **Prueba del sistema** (capítulo 9). El objetivo de esta fase se especifica la integración del software, se hace la integración del sistema y se prueba. Se prueba que el sistema cumpla con sus requerimientos y se completan los manuales.

Agenda y minuta de la reunión semanal

La agenda que deberá tener como puntos importantes:
- Discutir los reportes de la integración del sistema y de las pruebas de integración en la forma de Registro de defectos.
- Discutir el informe de la prueba del software.
- Discutir la creación de los manuales.
- Asuntos generales.

Fase de Cierre

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso, del estado de la configuración.

Actividades
- Dirigir la evaluación del equipo y personal. Es responsable de hacer esta evaluación.
- Coordinar la identificación de las lecciones aprendidas y sugerencias de mejoras. Es responsable de este documento.
- Participar en la entrega del producto.

Productos del que es responsable
- Evaluación del equipo y personal.
- Lecciones aprendidas y sugerencias de mejora.

Agenda y minuta de la reunión semanal

La agenda que deberá tener como puntos importantes:
- Discutir la evaluación del equipo y personal.
- Discutir las lecciones aprendidas y sugerencias de mejora.
- Discutir el informe de mediciones.
- Discutir el informe del estado de la configuración del sistema.
- Asuntos generales.

Capítulo 12

Administrador de desarrollo

12.1 Objetivos, habilidades y responsabilidades

Objetivos

- Entender el proceso de desarrollo.
- Dirigir al equipo en las actividades de desarrollo en cada una de las fases.

Habilidades

- Experiencia en programación.
- Conocimientos de lenguajes de programación, ambientes de programación, herramientas de apoyo.
- Reconocimiento del equipo por sus habilidades técnicas.

Responsabilidades

- Motiva y guía al equipo a seguir el proceso de desarrollo en cada una de sus fases.
- Es responsable por producir el producto de calidad.
- Usa y aprovecha al máximo las habilidades y los conocimientos de los miembros del equipo.

12.2 Actividades del rol a realizar todas las semanas

- Coordinar el desarrollo del producto.

12.3 Actividades específicas del rol en cada fase

Fase de Lanzamiento

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Actividades

- Dirigir al equipo en la definición de los objetivos del producto.
- Definir sus objetivos como miembro del equipo.

Fase de Estrategia

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo ciclo en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Actividades

- Entender los criterios para la estrategia y guiar la discusión.
- Documentar la estrategia seleccionada en la forma Estrategia. Ver 4.2.

Productos de los que es responsable
- **Forma Estrategia.**

Fase de Planeación

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta misma fase se hace el introduce el concepto de calidad y se introducen las *revisiones entre colegas* como técnica de calidad que se harán a lo largo del desarrollo.

Fase de Especificación de requerimientos

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado que se establecen en el Plan de pruebas del sistema.

Actividades
- Dirigir el trabajo de definición de requerimientos vía casos de uso e integrarlos como documento Especificación de requerimientos del software.
- Dirigir la creación del Plan de pruebas del software.

Productos de los que es responsable
- Texto con la definición del problema.
- Glosario de términos.
- Especificación de requerimientos del software que incluye:
    - Diagrama general de casos de uso.
    - Detalle de casos de uso.
    - Prototipo de la interfaz de usuario.
    - Lista de requerimientos no funcionales.
- Plan de pruebas del software.

Fase de Diseño

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración y establecer las clases con que se construirá el software en sus vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

Actividades

- Entender los principios del diseño para trabajo en equipo.
- Identificar la arquitectura adecuada al software.
- Dirigir la creación del Plan de prueba de integración e integrarlo.
- Identificar las clases importantes para el desarrollo y hacer los diagramas de clases para los casos de uso. Modelar la dinámica de dichas clases en la realización de los casos de uso.

Productos de los que es responsable
- Arquitectura con diagrama de paquetes
- Diagrama de distribución
- Plan de prueba de integración
- Diagramas de clases
- Diagramas de secuencia
- Diagrama de estados
- Plan de prueba de integración

Fase de Construcción

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

Actividades:
- Refinar los diagramas de clases hasta el nivel necesario para hacer la codificación.
- Construcción del código de los paquetes que le corresponden.
- Hacer el plan de pruebas unitarias.
- Hacer pruebas unitarias de caja blanca.
- Hacer pruebas unitarias de caja negra.

Productos de los que es responsable:
- Diagrama de clase con todos los atributos y sus tipo, métodos, sus parámetros y su resultado y seudocódigo para métodos complejos.
- Coordinar la codificación de todos los paquetes.
- Plan de prueba unitaria de cada unidad.

Fase de Prueba del sistema

- **Prueba del sistema** (capítulo 9). El objetivo de esta fase se especifica la integración del software, se hace la integración del sistema y se prueba. Se prueba que el sistema cumpla con sus requerimientos y se completan los manuales.

Actividades
- Hacer la especificación de la integración y prueba del sistema.

- Revisar el plan de integración que se desarrollo en la fase de diseño, completarlo según la implementación, realizar la integración y probarla.
- Revisar el plan de prueba del software realizado en la fase de requerimientos, actualizarlo según la implementación y efectuar las pruebas del sistema.
- Hacer los manuales del sistema.

Productos de los que es responsable:
- Especificación de la integración del sistema.
- Reporte de las pruebas de integración en las formas de Registro de defectos.
- Prueba del sistema
- Informe de la prueba del sistema.
- Manuales del sistema.

Fase de Cierre

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso, del estado de la configuración.

Actividades
- Participar en los informes del sistema.
- Coordinar la identificación de lecciones aprendidas y propuestas de mejora del proceso.
- Entregar el sistema al instructor.

Producto del que es responsable
- Sistema funcionando.
- Forma de lecciones aprendidas y propuestas de mejora.

Capítulo 13

Administrador de planeación

13.1 Objetivos, habilidades y responsabilidades

Objetivos

- Producir un plan para el equipo y para cada miembro del mismo que sea completo, preciso y adecuado.
- Dar seguimiento al plan cada semana.
- Dar seguimiento y detectar los riesgos que se presenten en el proyecto.

Habilidades

- Persona muy organizada, planeadora de sus actividades.
- Aún informalmente planea sus actividades y cree que planear es positivo.
- Reconocimiento del equipo por su organización.
- No es impositiva sino conciliadora, pero intolerante a faltas de compromiso.

Responsabilidades

- Produce un plan completo, preciso y adecuado para todo el equipo.
- Da seguimiento cada semana del estatus del plan.
- Registra riesgos que se presenten en la forma de Registro de riesgos y les da seguimiento en la forma Semana del equipo.

13.2 Actividades del rol a realizar todas las semanas

- Dar seguimiento a la planeación a través de la forma *Semana del equipo* que se llena a partir de las formas *Semana personales*

13.3 Actividades específicas del rol en cada fase

Fase de Lanzamiento

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Actividades

- Coordinar la identificación de riesgos en la forma de Registro de riesgos. En esta forma se registran los riesgos que se van detectando a lo largo del proyecto. Todos los miembros del equipo pueden registrar un riesgo al detectarlo.

Productos de los que es responsable

- **Forma de Registro de riesgos**

Fase de Estrategia

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo ciclo en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Fase de Planeación

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta misma fase se hace el introduce el concepto de calidad y se introducen las *revisiones entre colegas* como técnica de calidad que se harán a lo largo del desarrollo.

Actividades

- Dirigir la construcción del plan del equipo.

Productos de los que es responsable

- Plan del equipo.

Fase de Especificación de Requerimientos

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado que se establecen en el Plan de pruebas del software.

Fase de Diseño

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración y establecer las clases con que se construirá el software en sus vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

Fase de Construcción

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del

equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

Fase de Prueba del sistema

- **Prueba del sistema** (capítulo 9). El objetivo de esta fase se especifica la integración del software, se hace la integración del sistema y se prueba. Se prueba que el sistema cumpla con sus requerimientos y se completan los manuales.

Fase de Cierre

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso, del estado de la configuración.

Actividades

- Hacer el informe de mediciones.

Productos de los que es responsable

- Informe de mediciones.

Capítulo 14

Administrador de calidad

14.1 Objetivos, habilidades y responsabilidades

Objetivos

- Hacer que todos los miembros del equipo cumplan con el Plan de calidad.
- Conseguir que todos los miembros del equipo produzcan productos de calidad.
- Que se lleven a cabo las reuniones de revisión entre colegas según el Plan de calidad y se registren los defectos en la forma Registro de defectos.
- Hacer que se corrijan los defectos registrados.

Habilidades

- Persona ordenada e interesada en la calidad del software.
- Reconocimiento del equipo por su interés en la calidad.
- Saber hacer buenas revisiones a los productos.
- No tiene enemistades con otros miembros del equipo, lo que facilitará la organización de las reuniones de revisión.

Responsabilidades

- Coordina la definición de estándares y vigila que se cumplan.
- Hace un calendario para las revisiones entre colegas.
- Dirige las revisiones entre colegas y registran los defectos encontrados.
- Se asegura que se corrijan los defectos.

14.2 Actividades del rol a realizar todas las semanas

- Organizar la reunión de revisión entre colegas convocando al miembro que le toca participar cada semana.
- Hacer la lista de verificación de cada semana para la revisión de los productos de esa semana.
- Pedir que se corrijan los defectos registrados a los responsables de los productos con defectos.
- Entregar los productos revisados y corregidos al líder.

14.3 Actividades específicas del rol en cada fase

Fase de Lanzamiento

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Actividades

- Definir el estándar de los documentos de común acuerdo con el equipo. Para que exista uniformidad en los productos y todos tengan un formato semejante se debe definir el estándar que se seguirá.

Productos de los que es responsable

- **Estándar de documentación.**

Fase de Estrategia

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo ciclo en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Fase de Planeación

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta misma fase se hace el introduce el concepto de calidad y se introducen las *revisiones entre colegas* como técnica de calidad que se harán a lo largo del desarrollo.

Actividades

- Definir el calendario para las reuniones de revisión entre colegas. Este plan contiene el calendario de las reuniones de revisión entre colegas que se van a realizar a los productos de trabajo. En todas las reuniones participa el administrador de calidad que las coordina, pero cada semana algún miembro del equipo coopera en ellas. El plan consiste en una tabla con la fecha de la reunión y quién participa en esa semana en la reunión de revisión.
- Organizar la reunión de revisión entre colegas definiendo la lista de verificación para los productos de cada fase y llenando la forma de Registro de defectos.
- Entregar los productos revisados y corregidos al líder.

Productos de los que es responsable

- Calendario de las reuniones de revisión entre colegas.
- Lista de verificación.
- Forma de Registro de defectos de la revisión entre colegas de esa semana.

Fase de Especificación de requerimientos

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado que se establecen en el Plan de pruebas del software.

Fase de Diseño

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración y establecer las clases con que se construirá el software en sus vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

Fase de Construcción

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

Fase de Prueba del sistema

- **Prueba del sistema** (capítulo 9). El objetivo de esta fase se especifica la integración del software, se hace la integración del sistema y se prueba. Se prueba que el sistema cumpla con sus requerimientos y se completan los manuales.

Actividades

- Revisar el plan de integración que se desarrollo en la fase de diseño, completarlo según la implementación, realizar la implementación y probarla.
- Revisar el plan de prueba del sistema realizado en la fase de requerimientos, actualizarlo según la implementación y efectuar las pruebas del sistema.

Productos de los que es responsable

- Especificación de la integración y prueba del sistema.
- Reporte de las pruebas en la forma de Registro de defectos y/o tablas cruzadas
- Informe de la prueba del sistema.

Fase de Cierre

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso, del estado de la configuración.

Actividades

- Hacer el informe de las mediciones.

Productos de los que es responsable

- Informes de mediciones.

Capítulo 15

Administrador de apoyo

15.1 Objetivos, habilidades y responsabilidades

Objetivos

- Conseguir las herramientas necesarias para apoyar al equipo en su trabajo.
- No permitir que se hagan cambios no autorizados a productos ya aprobados.
- Facilitar reutilizar lo desarrollado en el primer ciclo para agilizar el segundo ciclo.

Habilidades

- Experto en la búsqueda de herramientas de apoyo al desarrollo.
- Experto en cómputo y entusiasmado por aprender y explicar al equipo nuevas herramientas.
- Disciplinado en el manejo de versiones de productos.
- Reconocimiento del equipo por su facilidad para encontrar y explicar nuevas herramientas.

Responsabilidades

- Proporciona al equipo las herramientas y métodos adecuados para su trabajo.
- Controla los cambios a los productos.
- Avisa a los desarrolladores cuando un cambio los afecte.
- Coordina las versiones del sistema.

15.2 Actividades del rol a realizar todas las semanas

- Resguardar la versión electrónica de los documentos y hacerlos disponibles a los miembros del equipo.
- Facilitar las herramientas que necesite el equipo para el proyecto.
- Llevar el control de cambios a los productos.

15.3 Actividades específicas del rol en cada fase

Fase de Lanzamiento

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Fase de Estrategia

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo

ciclo en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Actividades

- Crear los depósitos de los documentos en papel y electrónicos. Todos los productos que se van generando se almacenan en dos carpetas, una electrónica a la que deberán tener acceso todos los miembros del equipo y otra en papel para la revisión del instructor.
- En el segundo ciclo, definir el Plan de configuración. El plan de configuración se efectúa en el segundo ciclo. Contiene los documentos a los que se les lleva el control de cambios y cómo se hacen.

Productos de los que es responsable

- **Carpeta del equipo (impresa y electrónica)**
- **Plan de configuración**

Fase de Planeación

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta misma fase se hace el introduce el concepto de calidad y se introducen las *revisiones entre colegas* como técnica de calidad que se harán a lo largo del desarrollo.

Actividades

- Facilitar las herramientas para construir los diagramas de Gantt.

Fase de Especificación de requerimientos

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado que se establecen en el Plan de pruebas del software.

Actividades

- Facilitar las herramientas para construir los diagramas de casos de uso y el prototipo de la interfaz de usuario.

Fase de Diseño

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración y establecer las clases con que se construirá el software en sus vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

Actividades

- Facilitar las herramientas para diagramar los modelos del diseño.

Fase de Construcción

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

Actividades
- Facilitar las herramientas para la implementación y pruebas unitarias.

Fase de Prueba del sistema

- **Prueba del sistema** (capítulo 9). El objetivo de esta fase se especifica la integración del software, se hace la integración del sistema y se prueba. Se prueba que el sistema cumpla con sus requerimientos y se completan los manuales.

Fase de Cierre

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso, del estado de la configuración.

Actividades
- Hacer el informe del estado de la configuración.

Productos de los que es responsable
- Forma de informe del estado de la configuración.

Capítulo 16

Ingeniero de desarrollo

16.1 Objetivos, habilidades y responsabilidades

Objetivos

- Participar en el desarrollo de un sistema siguiendo todas sus fases.

Habilidades

- Tener conocimientos y experiencia en programación, estructuras de datos y bases de datos.

Responsabilidades

- Llena la forma Semana personal.
- Asiste a la reunión del equipo semanal.
- Participa en las reuniones de revisión entre colegas según el Plan de calidad.
- Revisa y corrige los productos de los que es responsable.
- Se ajusta a los estándares del equipo
- Genera productos de calidad.

16.2 Actividades del rol a realizar todas las semanas

- Llenar la forma Semana personal.
- Asistir a la reunión del equipo semanal.
- Participar en las reuniones de revisión entre colegas según el calendario.
- Revisar y corregir los productos de que es responsable.
- Ajustarse a los estándares del equipo al generar sus productos.

16.3 Actividades específicas del rol en cada fase

Fase de Lanzamiento

- **Lanzamiento** (capítulo 3). Los objetivos de la fase de Lanzamiento en el ciclo 1 son: organizar el equipo para el proyecto, definir sus objetivos, establecer los estándares para los documentos e identificar los riesgos a los que se puede enfrentar el proyecto.

Actividades

- Definir los objetivos de su rol.
- Participar en la definición de los estándares de documentación.
- Participar en la identificación de riesgos.

Fase de Estrategia

- **Estrategia** (capítulo 4). El objetivo de la fase de Estrategia en el ciclo 1 es plantear y elegir una estrategia para dividir el alcance del proyecto en dos ciclos. En el segundo ciclo en esta fase se introducen los conceptos básicos de la Administración de la configuración y la necesidad de controlar los cambios a los documentos generados en el primer ciclo.

Actividades

- Participar en la definición de la estrategia.

Fase de Planeación

- **Planeación** (capítulo 5). El objetivo de esta fase es hacer la planeación de las actividades del equipo indicando qué actividades se harán, cuándo y por quién. En esta misma fase se hace el introduce el concepto de calidad y se introducen las *revisiones entre colegas* como técnica de calidad que se harán a lo largo del desarrollo.

Actividades

- Participar en la definición del plan del equipo.

Fase de Especificación de requerimientos

- **Especificación de requerimientos** (capítulo 6). El objetivo de esta fase es iniciar la construcción propiamente dicha del producto. Por esta razón es importante entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado que se establecen en el Plan de pruebas del software.

Actividades

- Participar en la especificación de requerimientos.
  - Construir diagrama de casos de uso.
  - Hacer el glosario.
  - Detallar los casos de uso.
  - Hacer el prototipo del sistema.
  - Definir los requerimientos no funcionales.
  - Hacer el Plan de prueba del sistema.

Fase de Diseño

- **Diseño** (capítulo 7). Los objetivos de esta fase son: definir el proceso del diseño para el trabajo en equipo, describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, hacer el Plan de pruebas de integración y establecer las clases con que se construirá el software en sus vistas estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

Actividades

- Participar en la definición de diseño
  - Definir el estándar del diseño
  - Hacer los diagrama de paquetes y de distribución
  - Hacer los diagrama de clases
  - Establecer la dinámica con diagramas de secuencia
  - Definir la navegación con diagrama de estados
  - Producir el Plan para la prueba de integración.

Fase de Construcción

- **Construcción** (capítulo 8). El objetivo de esta fase es hacer la implementación del software. Se considera como parte de esta fase, el diseño detallado, la programación y las pruebas unitarias. Hasta la fase de diseño se requirió del trabajo y discusión del equipo completo, en esta fase se reparten las partes del sistema a cada ingeniero que se responsabiliza de entregar el código probado de las unidades que se le asignan.

Actividades

- Participar en la codificación y pruebas unitarias.
  - Hacer el diseño detallado de la parte que le corresponda.
  - Refinar los diagramas de paquetes que le correspondan.
  - Refinar los diagramas de clase que le correspondan.
  - Generar el código que le corresponda.
  - Hacer el Plan de pruebas unitarias y efectuar las pruebas que le correspondan.

Fase de Prueba del sistema

- **Prueba del sistema** (capítulo 9). El objetivo de esta fase se especifica la integración del software, se hace la integración del sistema y se prueba. Se prueba que el sistema cumpla con sus requerimientos y se completan los manuales.

Actividades

- Participar en la integración y prueba del sistema.
- Aplicar las pruebas de integración y sistema para el ciclo 1.
- Participar en la generación de manuales que se le asignen.

Fase de Cierre

- **Cierre** (capítulo 10). El objetivo de esta fase es definir las actividades para cerrar el ciclo de desarrollo. Se evalúa el desempeño personal del equipo, se registran las lecciones aprendidas y las sugerencias de mejora, se hacen los informes de mediciones del proceso, del estado de la configuración.

Actividades

- Participar en la evaluación del equipo y personal
- Participar en la identificación de lecciones aprendidas y sugerencias de mejora.

Apéndice A

Guión general para el curso

A.1 Objetivo del curso:

Aprender las mejores prácticas de la Ingeniería de Software para el trabajo en equipo.

1. Aprender los fundamentos y las prácticas de la Ingeniería de Software.
2. Aprender a trabajar en equipo.
3. Desarrollar un producto de software real de tamaño mediano en 2 ciclos con un subproducto funcional en cada ciclo.
4. Documentar las fases del proceso de desarrollo en cada ciclo.
5. Aprender a recolectar métricas de tiempo, tamaño y defectos en el trabajo personal y de equipo.

Las condiciones necesarias a cumplir para alcanzar esos objetivos son:

- Los alumnos tienen conocimientos de programación en un lenguaje orientado a objetos, estructuras de datos y bases de datos.
- Se tiene un laboratorio de cómputo con impresora y herramientas como editor de texto, para planeación, diagramación de UML, compilador para el lenguaje y un repositorio compartido.
- Se tiene un ayudante para la realización de las prácticas.
- Se tiene material didáctico disponible en un sitio web del curso.

Al final del curso se espera:

- Se hayan aprendido las prácticas de la Ingeniería de Software.
- Se haya trabajado como un equipo exitoso con roles.
- Se haya desarrollado un sistema de manera incremental en dos ciclos.
- Se haya generado una carpeta del proyecto que incluye la documentación completa y consistente.

A.2 Calendario de las fases y actividades a realizar en cada semana del curso.

El curso tiene una duración de 17 semanas de 5 horas a la semana, tres de clase teórica y 2 de práctica.

Cada semana se cumple con al menos una fase del proceso de software. Algunas fases requieren de 2 semanas.

En el segundo ciclo se cubren varias fases por semana.

Semana 1 Introducción a la Ingeniería de Software

- Definición de Ingeniería de Software.
- Software, su naturaleza y atributos.
- Principios de la Ingeniería de Software.
- Proceso de software.
  - Hacer la práctica 1 Introducción a la Ingeniería de software.
  - Leer el capítulo 1.

Semana 2 Desarrollo de software en equipo

- Principios del trabajo en equipo a través de roles.
- Formación de equipos y asignación de roles.
- Reuniones semanales.
- Mediciones y su papel en la Ingeniería de Software.
- Formas Semana personal y del equipo.
  - Hacer las prácticas 2 **Trabajo en equipo y 3 Formación del equipo**.
  - Leer el capítulo 2 y de los capítulos de los roles (capítulos del 11 al 15), las habilidades y responsabilidades generales de los roles.

Semana 3 Fase de Lanzamiento 1

- Definir de los objetivos.
- Establecer el estándar de documentación.
- Identificar los riesgos del proyecto.
  - **Hacer la práctica 4 Llenado de formas.**
  - **Hacer práctica 5 Lanzamiento**
  - Leer el capítulo 3 del libro.

Semana 4 Fase de Estrategia 1

- Analizar el problema y seleccionar la estrategia para resolverlo en dos ciclos.
- **Hacer la práctica 6 Estrategia.**
  - Leer el capítulo 4.

Fase de Planeación 1

- Presentación del proceso de desarrollo a seguir.
- Establecer el plan de equipo para el primer ciclo.
- Planear las revisiones entre colegas.
  - **Hacer la práctica 7 Planeación**.
  - Leer el capítulo 5.

Semana 5 Fase de Especificación de requerimientos 1

- Introducción a UML.
- Construir el diagrama de casos de uso.
- Casos de uso detallados.
- Planteamiento del prototipo de la interfaz del sistema.
  - **Hacer la práctica 8 Casos de uso.**
  - **Hacer la práctica 9 Prototipo**
  - Leer el capítulo 6.

Semana 6 Fase de Especificación de requerimientos 1

- Establecer los requerimientos no funcionales.
- Proponer el Plan de prueba del software.
  - **Hacer la práctica 10 Requerimientos no funcionales y plan de prueba del sistema**.

Semana 7 Fase de Diseño 1

- **Especificar el ambiente de implementación.**
- **Estándar del diseño**
- **Definición de la arquitectura del sistema con diagrama de paquetes**
- **Hacer el diagrama de distribución del sistema.**
- **Hacer la práctica 11 Arquitectura.**
  - Leer el capítulo 7.

Semana 8 Fase de Diseño 1

- **Construir de los diagramas de clases**
- **Construir los diagramas de secuencia**
- **Diseñar la navegación en la interfaz.**
- **Hacer el Plan de pruebas de integración**
- **Hacer la práctica 12 Diagramas de clases y secuencia**
- Leer el capítulo 7.

Semana 9 Fase de Construcción 1

- Diseñar la base de datos.
- Construir de componentes.
- Hacer las pruebas unitarias.
  - Leer el capítulo 8.

Semana 10 Fase de Construcción 1

- Construir las clases.
- Hacer las pruebas unitarias a las clases y componentes.
- **Hacer la práctica 13 Pruebas unitarias.**
  - Leer el capítulo 8.

Semana 11 Fase de Prueba del sistema 1

- Aplicar las pruebas de integración y sistema.
- Producir los manuales.
- **Hacer la práctica 14 Pruebas de integración y del sistema**
- **Hacer la práctica 15 Manuales.**
  - Leer el capítulo 9.

Semana 12 Fase de Cierre 1
- Evaluación personal y del equipos para el ciclo 1.
- Identificar las lecciones aprendidas y sugerencias de mejora.
- Hacer informes del sistema
- Entregar la primera versión del sistema y carpeta completa.
- **Hacer la práctica 16 Cierre.**
  - Leer el capítulo 10.
  - **Primer examen.**

Entrega del sistema del primer ciclo

Semana 13 Fases de Lanzamiento, Estrategia y Planeación 2
•Redefinir los objetivos para el segundo ciclo.
•Identificar los riesgos de este ciclo.
•Planear la administración de la configuración.
Revisar la consistencia de carpeta y sistema en la ayudantía

Semana 14 Fases de Especificación de Requerimientos y Diseño 2
•Ampliar los requerimientos y el plan de pruebas para el ciclo 2.
•Ampliar el prototipo de interfaz.
•Revisar la arquitectura y los componentes para el ciclo 2.
•Ampliar los diagramas de clases y secuencia.
•Ampliar el diseño de la base de datos
•Hacer la práctica 17 Administración de la Configuración del Software.

Semana 15 Fases de Construcción 2
- Construir los componentes y clases.
- Aplicar las pruebas de unitarias para el ciclo 2.

Semana 16 Fase de Prueba del sistema 2
- Aplicar las pruebas de integración.
- Aplicar las pruebas del sistema.
- Reflexionar sobre el curso y lo aprendido.

Semana 17 Fase de Cierre 2
- Entregar la segunda versión del sistema y carpeta completa.
- **Segundo examen.**

Entrega del sistema del segundo ciclo

Apéndice B

Formas para el curso

B.1 Forma Semana personal

B.2 Forma Semana del equipo

B.3 Forma de Registro de riesgos

B. 4 Forma Estrategia

B.5 Forma Registro de defectos

B.6 Forma de Evaluación del equipo y personal

B.7 Forma de Lecciones aprendidas y sugerencia de mejoras

B.8 Forma de Informe de mediciones

B.9 Forma de Informe del estado de la configuración

B.10 Forma Semana del equipo ciclo 2

B.11 Forma de Solicitud de cambio

B12 Forma Semanal del estado de los cambios

Forma Semana personal

Nombre:	**Equipo:**	**Semana:**	**Ciclo:**

Productos de desarrollo generados	**Tiempo dedicado**	**Tamaño del producto**

Forma Semana del equipo

Equipo	Semana	Ciclo

Productos de desarrollo generados esta semana	Tiempo dedicado	Tamaño del producto

Datos globales del proyecto en esta semana	Tiempo
Tiempo dedicado al proyecto en esta semana	
Tiempo dedicado al proyecto en este ciclo hasta esta semana	
Productos terminados de esta fase en esta semana	

Seguimiento de riesgos	Estado

Forma de Registro de riesgos

Equipo	**Semana**	**Ciclo**

Riesgo	**Probabilidad de ocurrencia**	**Impacto**	**Estrategia para manejar el riesgo**

Forma Estrategia

Equipo	**Ciclo**

Gráfica de Dependencias

Justificación de la estrategia

Funcionalidades o necesidades	**Ciclo 1 o 2**	

Forma Registro de defectos

Equipo	**Semana**	**Ciclo**

Producto:	**Fecha:**
Descripción de los defectos:	
•	
•	
•	

Producto:	**Fecha:**
Descripción de los defectos:	
•	
•	
•	

Producto:	**Fecha:**
Descripción de los defectos:	
•	
•	
•	

Total de productos revisados:	**Total de defectos encontrados:**

Forma de Evaluación del equipo y personal

Equipo	**Ciclo**

Evaluación del equipo:

Evaluación del equipo sobre los siguientes criterios. Encierre un número desde el 1 (bajo) hasta el 5 (alto)					
Espíritu de equipo	1	2	3	4	5
Efectividad del equipo	1	2	3	4	5
Experiencia adquirida	1	2	3	4	5
Productividad del equipo	1	2	3	4	5
Calidad del proceso	1	2	3	4	5
Calidad del producto	1	2	3	4	5

Evaluación de cada rol:

Para cada rol, evalúe el porcentaje de trabajo requerido y la dificultad durante este ciclo		
Rol	Trabajo requerido	Dificultad en el rol
Líder de Equipo		
Administrador de Desarrollo		
Administrador de Planeación		
Administrador de Calidad y Proceso		
Administrador de Apoyo		
Contribución total (100%)		

Evalúe la contribución de cada rol. Encierre un número desde el 1 (bajo) hasta el 5 (alto)					
Líder de Equipo	1	2	3	4	5
Administrador de Desarrollo	1	2	3	4	5
Administrador de Planeación	1	2	3	4	5
Administrador de Calidad y Proceso	1	2	3	4	5
Administrador de Apoyo	1	2	3	4	5

Evalúe cada rol de acuerdo al apoyo y ayuda proporcionada. Encierre un número desde el 1 (bajo) hasta el 5 (alto)					
Líder de Equipo	1	2	3	4	5
Administrador de Desarrollo	1	2	3	4	5
Administrador de Planeación	1	2	3	4	5
Administrador de Calidad y Proceso	1	2	3	4	5
Administrador de Apoyo	1	2	3	4	5

Evalúe cada rol de acuerdo a su desempeño. Encierre un número desde el 1 (bajo) hasta el 5 (alto)					
Líder de Equipo	1	2	3	4	5
Administrador de Desarrollo	1	2	3	4	5
Administrador de Planeación	1	2	3	4	5
Administrador de Calidad y Proceso	1	2	3	4	5
Administrador de Apoyo	1	2	3	4	5

Forma de Lecciones aprendidas y sugerencias de mejoras

Equipo	Ciclo

Mejores prácticas

Del equipo	**Del proceso**

Problemas recurrentes

Del equipo	**Del proceso**

Experiencias exitosas

Del equipo	**Del proceso**

Comentarios

Sugerencias de mejora

Para el proceso de desarrollo (métodos, herramientas, formas, estándares, etc.)

Mejoras al equipo de desarrollo.

Forma de Informe de mediciones

Equipo	**Ciclo**

Tiempos por fase

Fases	**Resumen de tiempos**
Lanzamiento	
Estrategia	
Planeación	
Especificación de requerimientos	
Diseño	
Construcción	
Integración y prueba del sistema	
Tiempo total del ciclo	

Tamaños de los productos

Productos	**Tamaños**	**Unidades**

Defectos encontrados por producto

Producto	**Número de Defectos**

Total de defectos	

Productividad y calidad de los productos

Producto	**Tamaño del código / tiempo total de desarrollo**	**Tamaño de producto / número de defectos**

Forma de Informe del estado de la configuración.

Equipo	**Ciclo**

Elementos de la configuración al final del ciclo:

Elemento de la configuración	**Versión**	**Dirección donde se encuentra**

Forma Semana del equipo ciclo 2

Equipo	**Semana**	**Ciclo**

Productos de desarrollo generados	**Tiempo estimado**	**Tiempo actual**	**Tamaño estimado**	**Tamaño actual**

Datos globales del proyecto en esta semana	**Tiempo**
Tiempo dedicado al proyecto en esta semana	
Tiempo dedicado al proyecto en este ciclo hasta esta semana	
Productos terminados de esta fase en esta semana	

Tiempo ganado en esta semana	
Tiempo ganado en este ciclo a la fecha	

Seguimiento de asuntos o riesgos	**Estado**

Forma de Solicitud de cambio

Equipo	Semana	Ciclo

Información del componente

Nombre_____Responsable_____
Dirección del respaldo_____

Razones para hacer el cambio

_____
_____
_____

Impactos del cambio

_____
_____
_____
_____

Descripción del cambio

_____
_____
_____
_____

Estado del cambio

□ Aprobado □ Rechazado □ En proceso

Forma Semanal del estado de los cambios

Equipo	Semana	Ciclo

Informe de la actividad de control de cambios

Formas de Solicitud de cambios:	A lo largo del ciclo
Enviadas	
Aprobadas	
En proceso	
Rechazadas	
En reversa (cambio que debe deshacerse)	

Estado de los cambios en esta semana:

Elemento de la configuración	Versión	Estado de los cambios propuestos	Estado de la implementación

Apéndice C

Orientaciones prácticas para el profesor

C.1 Planteamiento del problema a resolver

Este curso es teórico-práctico, lo que implica que se explica a los alumnos las buenas prácticas de la Ingeniería de Software y las apliquen en un caso de estudio.

En este curso el cliente que plantea el problema es el instructor, esto es para evitar el trabajo que cuesta el entendimiento con un cliente real y centrar la enseñanza en el proceso de desarrollo. En un segundo curso se puede tener un cliente lo que beneficiaría por tener una experiencia más real.

El instructor plantea a los alumnos un texto especificando el problema a resolver que tendrá las siguientes características:

- Debe ser una aplicación real. No debe ser un problema trivial pero tampoco muy complejo.
- Debe poder obtenerse un producto funcionando en 10 semanas que es la duración del primer ciclo y después incrementar su funcionalidad en un segundo ciclo de 5 semanas.
- Debe ser fácil de entender por los alumnos que no sea necesario hacer una investigación sobre el contexto y el problema, que les sea natural en su ambiente.
- Debe requerir de un equipo de trabajo con conocimientos varios como bases de datos, interfaz gráfica, etc.

- Ejemplo de problemas planteados son:
    - **Sistema de administración escolar** con funcionalidades como: consultar planes de estudios, inscribirse revisando requisitos de las materias, listas de alumnos por materias, etc.
    - **Sistema de préstamos bibliotecarios** con funcionalidades como: consultar el acervo, registrarse como usuarios de una biblioteca, pedir libros para préstamo domiciliario, regresarlos, obtener una multa por retraso en la devolución, etc.
    - **Sistema para el departamento de servicios médicos de una Universidad,** se requiere de un sistema de software que apoye al personal administrativo (clientes del sistema) en el registro de consultas y citas de pacientes (alumnado de la comunidad universitaria). El servicio es exclusivamente para alumnos y sólo para aquellos que tienen inscripción vigente.
    - Sistema de encuestas, se requiere un sistema que apoye a un cliente a encuestar a una población dada sobre algunos temas de interés. El cliente

define los temas de la encuesta, los datos personales que le interesan, y las preguntas de la encuesta, así como las posibles opciones de respuesta. El sistema pedirá al encuestado los datos personales y le ofrece distintas opciones para que conteste sobre los temas de la encuesta. Además, el sistema podrá efectuar estadísticas directas sobre los temas de la encuesta, como cuántas personas eligieron cada opción de cada pregunta y qué porcentaje son de la población total.

La tecnología con que se deberá programar depende de la carrera y del plan de estudios, así como de los conocimientos previos de los alumnos. El objetivo es enseñar a los alumnos a trabajar en equipo, a seguir el proceso de desarrollo de software y a construir un producto de calidad, no promover una programación virtuosa.

C.2 Formación de los equipos

La formación de los equipos puede ser asignada por el profesor o dejar a la libre elección de los alumnos. Ambos enfoques tienen ventajas y desventajas.

La asignación por el profesor es adecuada cuando los alumnos están en semestres intermedios de la carrera y llevan el curso en 5º o 6º semestre. Esto les permite conocer nuevos compañeros y posiblemente formar equipos para otros cursos.

Si es un curso de semestre más avanzado como nuestro caso (es de 7º), generalmente ya tienen formados sus equipos por la convivencia en otras materias y se conocen suficientemente para saber sus capacidades y deficiencias. Por lo que se deja la formación del equipo a la libre elección de los alumnos.

Muchas veces nos hemos encontrado con la situación de que un grupo de alumnos no son escogidos por nadie, no se conocen y el resultado en general, ha sido que estos equipos fracasan en la integración del equipo. Nuestra interpretación es que son alumnos poco sociales o no dispuestos a trabajar en conjunto con otros compañeros.

Pero salvo esas excepciones la experiencia en la formación de los equipos ha sido muy exitosa.

C.3 Calificación de los equipos

La experiencia nos ha indicado que se establezca desde el principio que la calificación será individual pero basada en el buen desempeño del equipo.

La base de la calificación estará dada por lo obtenido por el equipo. Esta calificación está dada por la entrega oportuna y bien hecha de los productos de cada semana y el buen funcionamiento del software al final de cada ciclo. Un software que funcione

maravillosamente pero no se siguió el proceso tiene una base de calificación de 6, pues lo importante en este curso es aprender el proceso y el trabajo en equipo.

Si el software no funciona, pero se siguió el proceso y el equipo se integró, la base de la calificación puede ser 6 también.

Si el software funciona adecuadamente y se siguió el proceso y el equipo trabajó bien, la calificación es 10.

La calificación individual está basada en esa calificación del equipo, se pueden sumar o bajar puntos al desempeño personal, de manera que no todos los integrantes del equipo tienen la misma calificación. Los *héroes o superman* y los que no hacen su trabajo pueden salir perjudicados pues impiden el correcto funcionamiento del equipo. De estos comportamientos, el instructor se da cuenta en las reuniones con los líderes de los equipos.

La fórmula usada para la calificación es:

- 60% carpeta y sistema que funciona bien y trabajo adecuado del equipo
- 40% trabajo individual como son asistencias, exámenes y prácticas de laboratorio.