CAPITULO II

INGENIERIA DEL SOFTWARE ASISTIDA POR COMPUTADORA.

2.1. QUE SIGNIFICA CASE?

Presenta 3 características fundamentales: (1) una colección de herramientas útiles que ayudan en cada paso de la construcción de un producto; (2) una disposición organizada que permite hallar rápidamente las herramientas y permite utilizarlas con eficiencia: (3) una persona calificada que sabe cómo utilizar de forma efectiva las herramientas. Los ingenieros del software reconocen que necesitan herramientas más variadas (las herramientas manuales por sí mismas, no satisfacen los requisitos de los sistemas basados en computadora modernos).

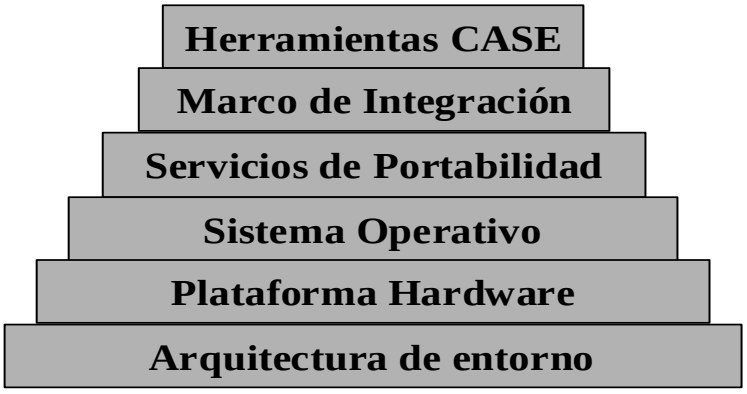
Las herramientas CASE son un complemento de la caja de herramientas del ingeniero del software. CASE proporciona al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería. Al igual que las herramientas de ingeniería y de diseño asistidos por computadora que utilizan los ingenieros de otras disciplinas, las herramientas CASE ayudan a asegurar que la calidad sea algo diseñado antes de llegar a construir el producto.

2.2. BLOQUES BASICOS DEL CASE

La ingeniería del software asistida por computadora puede ser tan simple como una única herramienta que permita desarrollar una actividad específica, o tan compleja como un "entorno" que integre distintas herramientas, una base de datos, personas, hardware, una red, sistemas operativos, estándares y muchos otros componentes. Los bloques de construcción de CASE se ilustran en la figura 2.1. Cada bloque de construcción forma un fundamento para el siguiente, estando las herramientas situadas en la parte superior del montón. Es interesante tener en cuenta que el fundamento de los entornos CASE efectivos tiene relativamente poco que ver con las herramientas de ingeniería del software en sí. Más bien, los entornos que tienen éxito para la ingeniería del software se construyen basándose

en una *arquitectura de entorno* que abarca un hardware adecuado y un software de sistema adecuado. Además, la arquitectura de entorno debe considerar los patrones de trabajo humanos que se aplican durante el proceso de ingeniería del software.

Figura. 2.1



La arquitectura del entorno, que constan de una plataforma hardware y de un apoyo de sistema operativo (incluyendo el software de red y de gestión de la base de datos), constituye los fundamentos CASE. Pero el entorno CASE en sí requiere otros bloques de construcción. Existe un conjunto de servicios de portabilidad que proporciona un puente entre las herramientas CASE y su *marco de referencia de integración* y la arquitectura del entorno. El marco de referencia de integración es una colección de programas especializados que capacitan a las herramientas CASE individuales para comunicarse entre sí, para crear una base de datos del proyecto y para mostrar el mismo aspecto al usuario final (el ingeniero del software). Los servicios de portabilidad permiten que las herramientas CASE y su marco de referencia de integración migren entre distintas plataformas del hardware y sistemas operativos sin un mantenimiento adaptativo que resulte significativo.

Los bloques de construcción representados en la figura 2.1 representan un fundamento exhaustivo para la integración de herramientas CASE. Sin embargo, la mayor parte de las herramientas CASE que se utilizan en la actualidad no han sido construidas empleando todos los bloques de construcción que se describen más arriba. De hecho, algunas

herramientas CASE siguen siendo *soluciones puntuales*. Esto es, se utiliza una herramienta para que preste su apoyo en una actividad de ingeniería del software concreta, pero esta herramienta no se comunica directamente con otras, no está unida a una base de datos del proyecto y no forma parte de un entorno integrado CASE (I-CASE). Aun cuando esta situación no es ideal, se puede utilizar una herramienta CASE bastante eficiente, aunque se trate de una solución puntual.

2.3 UNA TAXONOMIA DE HERRAMIENTAS CASE.

Existe un cierto número de riesgos que son inherentes siempre que se intenta efectuar una categorización de las herramientas CASE. Existe una sutil implicación consistente en que para crear un entorno CASE efectivo uno debe de implementar todas las categorías de herramientas. Se puede crear una confusión al ubicar una herramienta específica dentro de una categoría, cuando otras personas pueden creer que pertenece a otra categoría.

Las herramientas CASE se pueden clasificar por su función, por su papel como instrumentos para administradores o personal técnico, por su utilización en los distintos pasos del proceso de ingeniería del software, por la arquitectura de entorno (hardware y software) que les presta su apoyo, o incluso por su origen o su coste. La taxonomía que se presenta más abajo utiliza como criterio principal la función.

Herramientas de la ingeniería de la información. Al modelar los requisitos de información estratégica de una organización, las *herramientas de la ingeniería de la información* proporcionan un **metamodelo** del cual se derivan sistemas de información específicos. En lugar de centrarse en los requisitos de una aplicación específica, estas herramientas CASE modelan la información de negocios cuando ésta se transfiere entre distintas entidades organizativas en el seno de una compañía. El objetivo primordial de las herramientas de esta categoría consiste en representar objetos de datos de negocios, sus relaciones, y la forma en que fluyen estos objetos de datos entre distintas zonas de negocio en el seno de la compañía.

Modelado de procesos y herramientas de administración. Si una organización intenta mejorar un proceso de negocios (o de software) lo primero que debe de hacer es entenderlo. Las herramientas de modelado de procesos se utilizan para representar los elementos clave del proceso de modo que sea posible entenderlo mejor. Estas herramientas también pueden proporcionar vínculos con descripciones de procesos que ayuden a quienes estén implicados en el proceso de comprender las tareas que se requieren para llevar a cabo ese proceso. Además, las herramientas de administración de procesos pueden proporcionar vínculos con otras herramientas que proporcionen un apoyo para actividades de proceso ya definidas.

Herramientas de planificación de proyectos. Las herramientas de esta categoría se concentran en dos áreas primordiales: estimación de esfuerzos de proyecto y de costes de software, y planificación de proyectos. Las herramientas de planificación de proyectos capacitan al administrador para definir todas las tareas del proyecto (la estructura de desglose de tareas), para crear una red de tareas (normalmente empleando una entrada gráfica), para representar las interdependencias entre tareas y para modelar la cantidad de paralelismo que sea posible para ese proyecto.

Herramientas de análisis de riesgos. La identificación de riesgos potenciales y el desarrollo de un plan para mitigar, monitorizar y administrar esos riesgos tiene una importancia fundamental en los grandes proyectos. Las herramientas de análisis de riesgos capacitan al administrador del proyecto para construir una tabla de riesgos proporcionando una guía detallada en la identificación y análisis de riesgos.

Herramientas de administración de proyectos. La planificación del proyecto y el plan del proyecto deben de seguirse y de monitorizarse de forma continua. Además, el gestor deberá de utilizar las herramientas que recojan métricas que en última instancia proporcionen una indicación de la calidad del producto del software. Las herramientas de esta categoría suelen ser extensiones de herramientas de planificación de proyectos.

Herramientas de seguimiento de requisitos. Cuando se desarrollan grandes sistemas, el sistema proporcionado suele no satisfacer los requerimientos especificados por el cliente. El objetivo de las herramientas de seguimiento de requisitos es proporcionar un enfoque sistemático para el aislamiento de requisitos, comenzando por la solicitud del cliente de una propuesta (RFP) o especificación. Las herramientas de trazado de requisitos típicas combinan una evaluación de textos por interacción humana, con un sistema de gestión de bases de datos que almacena y categoriza todos y cada uno de los requisitos del sistema que se analizan a partir de la RFP o especificación original.

Herramientas de métricas y gestión. Las métricas de software mejoran la capacidad del administrador para controlar y coordinar el proceso del software y la capacidad del ingeniero para mejorar la calidad del software que se produce. Las métricas y herramientas de medida actuales se centran en procesos, proyectos y características del producto. Las herramientas orientadas técnicamente determinan métricas técnicas que proporcionan una mejor visión de la calidad del diseño o del código. Muchas de las herramientas métricas avanzadas mantienen una base de datos de medidas de medias de la industria. Basándose en características de proyectos y de productos proporcionados por el usuario, estas herramientas *califican* los números locales frente a los valores medios de la industria (y frente al rendimiento local anterior) y sugieren estrategias para llegar a mejoras.

Herramientas de documentación. Las herramientas de producción de documentos y autoedición prestan su apoyo a casi todos los aspectos de la ingeniería, y representan una importante oportunidad de aprovechamiento para todos los desarrolladores de software. La mayor parte de las organizaciones dedicadas al desarrollo de software invierte una cantidad de tiempo considerable en el desarrollo de documentos, y en muchos casos el proceso de documentación en sí resulta bastante deficiente. No es infrecuente que una organización de desarrollo de software invierta hasta un 20 o un 30 por ciento de su esfuerzo global de desarrollo de software en la documentación. Por esta razón, las herramientas de documentación suponen una oportunidad importante para mejorar la productividad.

Herramientas de software de sistema. CASE es una tecnología de estaciones de trabajo. Por tanto, el entorno CASE debe adaptarse a un software de sistema de red de alta calidad, al correo electrónico, a los boletines electrónicos y a otras capacidades de comunicaciones.

Herramientas de control de calidad. La mayor parte de las herramientas CASE que afirman que tienen como principal interés el control de calidad son en realidad herramientas métricas que hacen una auditoría del código fuente para determinar si se ajusta o no a ciertos estándares del lenguaje. Otras herramientas extraen métricas técnicas en un esfuerzo por extrapolar la calidad del software que se está construyendo.

Herramientas de gestión de base de datos. El software de gestión de bases de datos sirve como fundamento para establecer una base de datos CASE (depósito), que también se denominará base de datos del proyecto. Dado el énfasis acerca de los objetos de configuración, las herramientas de gestión de bases de datos para CASE pueden evolucionar a partir de los sistemas de gestión de bases de datos relacionales (SGBDR) para transformarse en sistemas de gestión de bases de datos orientadas a objetos (SGBDOO).

Herramientas de gestión de configuración de software. La gestión de configuración de software (GCS) se encuentra en el núcleo de todos los entornos CASE. Las herramientas pueden ofrecer su asistencia en las cinco tareas principales de GCS: identificación, control de versiones, control de cambios, auditoría y contabilidad de estados. La base de datos CASE proporciona un mecanismo para identificar todos los elementos de configuración y relacionarlo con otros elementos.

Herramientas de análisis y diseño. Las herramientas de análisis y diseño capacitan al ingeniero del software para crear modelos del sistema que haya que construir. Los modelos contienen una representación de los datos, de la función y del comportamiento (en el nivel de análisis), así como caracterizaciones del diseño de datos, arquitectura, procedimientos e interfaz. Al efectuar una comprobación de la consistencia y validez del modelo, las herramientas de análisis y diseño proporcionan al ingeniero de software un cierto grado de

visión en lo tocante a la representación del análisis, y le ayudan a eliminar errores antes de que se propaguen al diseño, o lo que es peor, a la propia implementación.

Herramientas PRO/SIM. Las herramientas PRO/SIM (de prototipos y simulación) proporcionan al ingeniero de software la capacidad de predecir el comportamiento de un sistema en tiempo real antes de llegar a construirlo. Además, capacitan al ingeniero del software para desarrollar simulaciones del sistema en tiempo real que permitirán al cliente obtener ideas acerca de su funcionamiento, comportamiento, y respuesta antes de la verdadera implementación.

Herramientas de desarrollo y diseño de interfaz. Las herramientas de desarrollo y diseño de interfaz son en realidad un conjunto de primitivas de componente de programas tales como menús, botones, estructuras de ventanas, iconos, mecanismos de desplazamiento, controladores de dispositivos etc. Sin embargo, estos conjuntos de herramientas se están viendo sustituidos por herramientas de generación de prototipos de interfaz que permiten una rápida creación en pantalla de sofisticadas interfaces de usuario, que se ajustan al estándar de interfaz que se haya adoptado para el software.

Herramientas de generación de prototipos. Se pueden utilizar toda una gama de herramientas de generación de prototipos. Los generadores de pantallas permiten al ingeniero de software definir rápidamente la disposición de la pantalla para aplicaciones interactivas. Otras herramientas de prototipos CASE más sofisticadas permiten la creación de un diseño de datos, acoplado con las disposiciones de la pantalla y de los informes simultáneamente. Muchas herramientas de análisis y diseño proporciona extensiones PRO/SIM generan un esqueleto de código fuente en Ada y C para las aplicaciones de ingeniería (en tiempo real). Por último, una gama de herramientas de cuarta generación poseen también características de generación de prototipos.

Herramientas de programación. La categoría de herramientas de programación abarca los compiladores, editores, y depuradores que están disponibles para prestar su apoyo en la

mayoría de los lenguajes de programación convencionales. Además, los entornos de programación orientados a objetos (OO), los lenguajes de cuarta generación, los entornos de programación gráfica, los generadores de aplicaciones, y los lenguajes de consulta de bases de datos residen también en esta categoría.

Herramientas de integración y comprobación. En su directorio de herramientas de comprobación de software, Software Quality Engineering define las siguientes categorías de herramientas de comprobación:

- *Adquisición de datos:* herramientas que adquieren datos que se utilizarán durante la comprobación.
- *Medida estática:* herramientas que analizan el código fuente sin ejecutar casos de prueba.
- *Medida dinámica:* herramientas que analizan el código fuente durante la ejecución.
- *Simulación:* herramientas que simulan las funciones del hardware o de otros elementos externos.
- *Administración de comprobaciones:* herramientas que prestan su asistencia en la planificación, desarrollo y control de comprobaciones.
- *Herramientas de funcionalidad cruzada:* se trata de herramientas que cruzan los límites de las categorías anteriores.

Debería tenerse en cuenta que muchas de las herramientas de comprobación poseen características que abarcan dos o más de las categorías anteriores.

Herramientas de análisis estático. Las herramientas de análisis estático prestan su asistencia al ingeniero del software a efectos de derivar casos prácticos. Se utilizan tres tipos distintos de herramientas estáticas de comprobación en la industria: Herramientas de comprobación basadas en código, lenguajes de comprobación especializados, y herramientas de comprobación basadas en requisitos. Las *herramientas de comprobación basadas en código* admiten un código fuente (o PDL) como entrada, y efectúan un cierto número de análisis que dan lugar a la generación de casos de prueba. Los *lenguajes de*

comprobación especializados capacitan el ingeniero del software para escribir detalladas especificaciones de comprobación que describirán todos los casos de prueba y la logística de su ejecución. Las herramientas de comprobación basadas en requisitos aíslan requisitos específicos del usuario y sugieren casos de prueba (o clases de comprobaciones) que ejerciten estos requisitos.

Herramientas de análisis dinámico. Las herramientas de análisis dinámico interactúan con un programa que se esté ejecutando, comprueban la cobertura de rutas, comprueban las afirmaciones acerca del valor de variables específicas y en general instrumentan el flujo de ejecución del programa. Las herramientas dinámicas pueden ser bien intrusivas, bien no intrusivas. Las *herramientas intrusivas* modifican el software que hay que comprobar mediante sondas que se insertan (instrucciones adicionales) y que efectúan las actividades mencionadas anteriormente. Las *herramientas de comprobación no intrusivas* utilizan un procesador hardware por separado que funciona en paralelo con el procesador que contenga el programa que se está comprobando.

Herramientas de gestión de comprobación. Las herramientas de gestión de comprobación se utilizan para comprobar y coordinar la comprobación de software para cada uno de los pasos principales de comprobación. Las herramientas de esta categoría administran y coordinan la comprobación de regresiones, efectúan comparaciones que determinan las diferencias entre la salida real y la esperada, y efectúan comprobaciones por lotes de programas con interfaces interactivas entre hombre y máquina. Además de las funciones indicadas anteriormente, muchas herramientas de gestión de comprobaciones sirven también como controladores de comprobación genéricos. Un controlador de comprobación lee uno o más casos de prueba de algún archivo de pruebas, da formato a los datos de prueba para que se ajusten a las necesidades del software que se está probando, e invoca entonces al software que sea preciso comprobar.

Herramientas de comprobación cliente/servidor. El entorno C/S exige unas herramientas de comprobación especializadas que ejerciten la interfaz gráfica de usuario y los requisitos de comunicaciones en red para el cliente y el servidor.

Herramientas de reingeniería. La categoría de herramientas de reingeniería se puede subdividir en las funciones siguientes:

- ***Herramientas de ingeniería inversa para producir especificaciones:*** se toma el código fuente como entrada y se generan modelos gráficos de análisis y diseño estructurados, listas de utilización y otras informaciones de diseño.
- ***Herramientas de reestructuración y análisis de código:*** se analiza la sintaxis del programa, se genera una gráfica de control de flujo y se genera automáticamente un programa estructurado; y
- ***Herramientas de reingeniería para sistemas en línea:*** se utilizan para modificar sistemas de bases de datos en línea (por ejemplo para convertir archivos IDMS o DB2 traduciéndolos a un formato de entidades y relaciones).

Muchas de las herramientas anteriores están limitadas a lenguajes de programación específicos (aun cuando se abarcan la mayoría de los lenguajes principales) y requieren un cierto grado de interacción con el ingeniero del software.

Las herramientas de ingeniería inversa y progresiva de la próxima generación harán un uso mucho mayor de técnicas de inteligencia artificial, aplicando una base de conocimientos que sea específica del dominio de la aplicación. El componente de inteligencia artificial asistirá en la descomposición y reconstrucción del sistema, pero seguirá requiriendo una interacción con un ingeniero de software a lo largo del ciclo de la reingeniería.

2.4. ENTORNOS CASE INTEGRADOS

Aun cuando se pueden obtener beneficios a partir de herramientas CASE individuales que abarquen actividades de ingeniería del software por separado, la verdadera potencia de

CASE solamente se puede lograr mediante la integración. Los beneficios del CASE *integrado* (I-CASE) incluyen (1) una transferencia suave de información (modelos, programas, documentos, datos) entre una herramienta y otra, y entre un paso de ingeniería y el siguiente; (2) una reducción del esfuerzo necesario para efectuar actividades globales tales como la administración de configuración de software, el control de calidad y la producción de documentos; (3) un aumento del control del proyecto que se logra mediante una mejor planificación, monitorización y comunicación; y (4) una mejor coordinación entre los miembros del personal que están trabajando en grandes proyectos de software.

Ahora bien, I-CASE también presenta desafíos significativos. En cada acción exige unas representaciones consistentes de la información de la ingeniería del software, unas interfaces estandarizadas entre herramientas, un mecanismo homogéneo para la comunicación entre el ingeniero de software y todas sus herramientas y un enfoque efectivo que capacite a I-CASE para desplazarse a lo largo de distintas plataformas de hardware y distintos sistemas operativos. Aun cuando las soluciones de los problemas implícitos en estos desafíos se han propuesto ya, los entornos I-CASE generales sólo empiezan a emerger en la actualidad.

El término *integración* implica tanto la *combinación* como el *cierre.* La I-CASE combina toda una gama de herramientas diferentes y de informaciones distintas de tal modo que hace posible el cierre de la comunicación entre herramientas, entre personas, y entre procesos de software. Se integran las herramientas de tal modo que la información de ingeniería del software esté disponible para todas las herramientas que se necesiten; la utilización se integra de tal modo que se proporcione un aspecto común para todas las herramientas; Y se integra una filosofía de desarrollo, implicando un enfoque de ingeniería del software estandarizado que aplique prácticas modernas y métodos ya probados.

Para definir la *integración* en el contexto del proceso del software, es necesario establecer un conjunto de requisitos para I-CASE. Un entorno CASE integrado debería de:

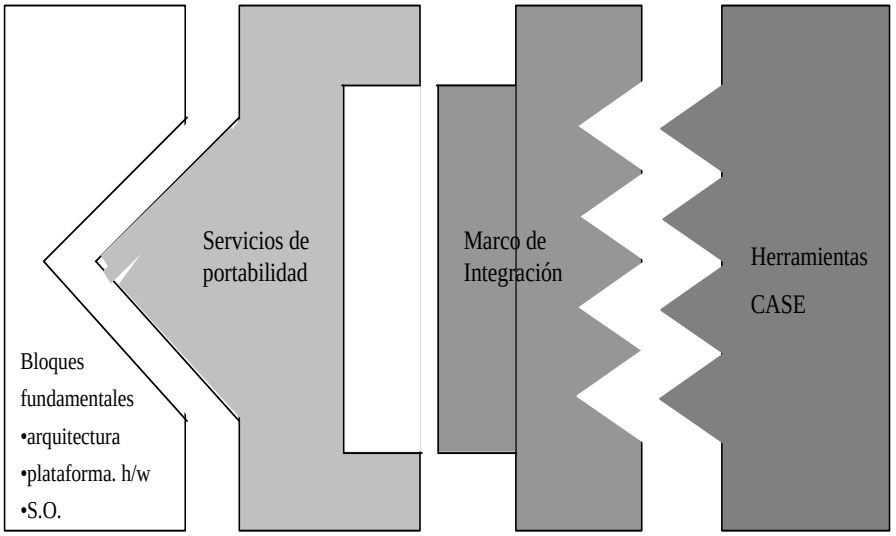
- Proporcionar un mecanismo para compartir la información de ingeniería del software entre todas las herramientas que estén contenidas en el entorno;
- Hacer posible que un cambio de un elemento de información se siga hasta los demás elementos de información relacionados;
- Proporcionar un control de versiones y una gestión de configuración general para toda la información de la ingeniería del software.
- Permitir un acceso directo y no secuencial a cualquiera de las herramientas contenidas en el entorno;
- Establecer un apoyo automatizado para un contexto de procedimientos para el trabajo de la ingeniería del software que integra las herramientas y los datos de una estructura de desglose de trabajo estandarizada.
- Capacitar a los usuarios de cada una de las herramientas para experimentar una utilización consistente en la interfaz hombre - máquina.
- Permitir la comunicación entre ingenieros del software; y
- Recoger métricas tanto de gestión como técnicas que se puedan utilizar para mejorar el proceso y el producto.

Para alcanzar estos objetivos, cada uno de los bloques de construcción de una arquitectura CASE (fig. 2.1) debe de encajar con los demás sin costuras. Según se muestra en la figura 2.2, los bloques de construcción fundamentales - arquitectura del entorno, plataforma hardware y sistema operativo - deben de *unirse* a través de un conjunto de servicios de portabilidad a un *marco de referencia de integración* que alcance los objetivos indicados anteriormente.

2.5. LA ARQUITECTURA DE INTEGRACION

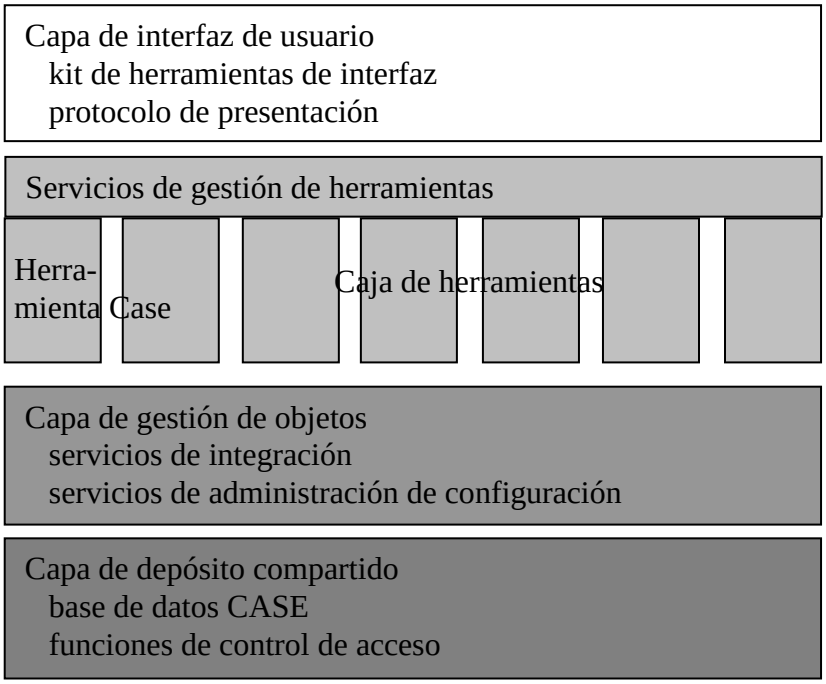
Un equipo de ingeniería del software utiliza herramientas CASE, los métodos correspondientes, y un marco de referencia de proceso con objeto de crear un conjunto de informaciones de ingeniería del software.

Figura. 2.2. Elementos de I-CASE



El marco de referencia de integración facilita la transferencia de información desde y hacia ese conjunto de informaciones. Para lograr esto, deben de existir los siguientes componentes de arquitectura: una base de datos para almacenar la información, un sistema de administración de objetos que gestione las cambios efectuados en la información, un mecanismo de control de herramientas que coordine la utilización de herramientas CASE, y una interfaz de usuario que proporcione una ruta consistente entre las acciones efectuadas por el usuario y las herramientas contenidas en el entorno. La mayoría de los modelos del marco de referencia de integración representan a estos componentes como si fueran capas. Se muestra en la figura 2.3 un modelo sencillo del marco de referencia que representa únicamente los componentes indicados más arriba.

Figura. 2.3



La capa de *interfaz de usuario* (Fig. 2.3) incorpora un conjunto de herramientas de interfaz estandarizado, con un protocolo de presentación común. El *kit de herramientas de interfaz* contiene software para la gestión de la interfaz hombre - máquina, y una biblioteca de objetos de visualización. Ambos proporcionan un mecanismo consistente para la comunicación entre la interfaz y las herramientas CASE individuales. El *protocolo de presentación* es el conjunto de líneas generales que proporciona a todas las herramientas CASE un mismo aspecto. Las convenciones de distribución de pantalla, los nombres de menú y la organización, los iconos, los nombres de los objetos, la utilización del teclado y el ratón y el mecanismo para acceder a las herramientas se definen todos ellos como parte del protocolo de presentación.

La *capa de herramientas* contiene un conjunto de servicios de gestión de herramientas con las herramientas CASE en sí. Los *servicios de gestión de herramientas (TMS)* controlan el comportamiento de las herramientas dentro del entorno. Si se emplea multitarea durante la ejecución de una o más herramientas, TMS efectúa la sincronización y comunicación multitarea, coordina el flujo de información desde el depósito y sistema de gestión de

objetos a las herramientas, realiza las funciones de seguridad y auditoría y recoge métricas acerca de la utilización de herramientas.

La *capa de gestión de objetos (OML)* lleva a cabo las funciones de gestión de configuración. En esencia, el software de esta capa de la arquitectura de marco de referencia proporciona el mecanismo para la integración de herramientas. Cada herramienta CASE se "enchufa" en la capa de gestión de objetos. Funcionando en conjunción con el depósito CASE, la OML proporciona los *servicios de integración* - un conjunto de módulos estándar que acoplan las herramientas con el depósito -. Además, la OML proporciona los servicios de gestión de configuración haciendo posible la identificación de todos los objetos de configuración, llevando a cabo el control de versiones, y proporcionando apoyo para el control de cambios, las auditorías y la contabilidad de estados.

La *capa de depósito compartido* es la base de datos CASE y las funciones de control de acceso que hacen posible que la capa de gestión de objetos interactúe con la base de datos. La integración de datos se logra mediante las capas de gestión de objetos y de depósito compartido.

2.6. EL DEPOSITO (REPOSITORY) CASE

Durante las primeras fases de la historia del desarrollo del software, el depósito era en realidad una persona – el programador que tenía que recordar la ubicación de toda la información relevante para un determinado proyecto de software, que tenía que recordar información que nunca se había escrito, y reconstruir información que se había perdido. Tristemente, el uso de una persona como el centro de acumulación y almacenamiento, no funciona demasiado bien. En la actualidad, el depósito es una "cosa", una base de datos que actúa como centro tanto para la acumulación como para el almacenamiento de información de ingeniería del software. El papel de la persona (del ingeniero del software) es interactuar con el depósito empleando herramientas CASE integradas con él.

2.6.1. El papel del depósito en I-CASE

El depósito de un entorno I-CASE es el conjunto de mecanismos y de estructuras de datos que consiguen la integración entre datos y herramientas y entre datos y datos. Proporciona las funciones y herramientas de un sistema de gestión de bases de datos, pero además, el depósito lleva a cabo o precipita las funciones siguientes:

- ***Integridad de datos:*** incluye funciones para validar las entradas efectuadas en el depósito, para asegurar la consistencia entre objetos relacionados, y para efectuar automáticamente modificaciones "en cascada" cuando un cambio efectuado en un objeto exige algún cambio en otros objetos relacionados con él.
- ***Información compartida:*** proporciona un mecanismo para compartir información entre múltiples desarrolladores y entre múltiples herramientas, gestiona el control multiusuario a los datos, y bloquea/desbloquea objetos para que los cambios no se superpongan inadvertidamente.
- ***Integración datos – herramientas:*** establece un modelo de datos al que pueden acceder todas las herramientas del entorno I-CASE, controla el acceso a los datos, y lleva a cabo las funciones de gestión de configuración adecuadas;
- ***Integración datos – datos:*** el sistema de gestión de bases de datos relaciona los objetos de datos de tal modo que se puedan llevar a cabo las demás funciones;
- ***Imposición de la metodología:*** el modelo E-R de datos almacenado en el depósito puede implicar un paradigma específico de ingeniería del software, como mínimo, las relaciones y los objetos definen un conjunto de pasos que será preciso realizar para construir el contenido del depósito; y
- ***Estandarización de documentos:*** la definición de objetos de la base de datos da lugar directamente a un enfoque estándar para la creación de documentos de ingeniería del software.

Para realizar estas funciones, se define el depósito en términos de un metamodelo. El metamodelo determina la forma en que se almacena la información en el depósito, la forma en que las herramientas pueden acceder a los datos y estos datos pueden ser visualizados

por los ingenieros de software, el grado hasta el cual se puede mantener la seguridad e integridad de los datos y la facilidad con que se puede extender el modelo ya existente para admitir nuevas necesidades.

El metamodelo es la plantilla en la cual se sitúa la información de ingeniería del software. Los metamodelos de entidades y relaciones se pueden crear también, pero también se están considerando otros modelos más sofisticados.

2.6.2. Características y contenidos.

Las características y contenido del depósito se entienden especialmente bien examinándolo desde dos perspectivas: ¿Qué es lo que hay que almacenar en el depósito, y qué servicios específicos son los que proporciona el depósito? En general, los tipos de cosas que habrá que almacenar en el depósito incluyen:

- El problema que hay que resolver.
- Información acerca del dominio del problema.
- La solución del sistema a medida que vaya surgiendo.
- Las reglas e instrucciones relativas al proceso de software (metodología) que se está siguiendo.
- El plan del proyecto, sus recursos y su historia.
- Información acerca del contexto organizativo.

En la tabla 2.1 se ha incluido una lista detallada de tipos de representaciones, documentos y productos que se almacenan en el depósito CASE.

Tabla 2.1: Contenido de un depósito CASE.

Información de la empresa	**Construcción**
Estructura organizativa	Código fuente; código objeto
Análisis de área de negocios	Instrucciones de construcción del sistema
Funciones de negocios	Imágenes binarias
Reglas de negocios	Dependencias de configuración
Modelos de procesos (escenarios)	Información de cambios
Arquitectura de la información.	**Validación y verificación**
Diseño de aplicaciones	Plan de comprobación; casos de prueba de los datos
Reglas de metodología	Guiones de comprobaciones de regresión
Representaciones gráficas	Resultados de comprobaciones
Diagramas de sistema	Análisis estadísticos
Estándares de denominación	Métricas de calidad de software
Reglas de integridad de referencia	**Información de gestión del proyecto**
Estructuras de datos	Planes de proyecto
Definiciones de proceso	Estructura de desglose de tareas
Definiciones de clase	Estimaciones; planes
Arboles de menú	Carga de recursos; informe de problemas
Criterios de rendimiento	Solicitudes de cambios; informes de estado
Restricciones temporales	Información de auditoría
Definiciones de pantalla	**Documentación del sistema**
Definiciones de informes	Documentos de requisitos
Definiciones lógicas	Diseños internos/externos
Lógicas de comportamiento	Manuales de usuario
Algoritmos	
Reglas de transformación	

Un depósito CASE robusto proporciona dos clases distintas de servicios: (1) los tipos de servicios que puede uno esperar de cualquier sistema de gestión de bases de datos sofisticado; y (2) servicios que son específicos de un entorno CASE.

Muchos requisitos del depósito son iguales a los de aplicaciones típicas que se construyen tomando como base un sistema de gestión de bases de datos comercial (SGBD). De hecho, muchos de los depósitos CASE actuales hacen uso de un SGBD (normalmente relacional u orientado a objetos) como tecnología de gestión de datos básica. Las características de un

SGBD estándar en un depósito CASE que preste su apoyo para la gestión de información de desarrollo del software incluye lo siguiente:

Almacenamiento de datos no redundantes. El depósito CASE proporciona un único lugar para el almacenamiento de toda la información pertinente al desarrollo de los sistemas de software, eliminando una duplicación que supone un desperdicio y es potencialmente proclive a errores.

Acceso de alto nivel. El depósito proporciona un mecanismo de acceso a los datos común de tal modo que no sea preciso duplicar las capacidades de gestión de datos en todas las herramientas CASE.

Independencia de datos. Las herramientas CASE de las aplicaciones blanco se aíslan del almacenamiento físico para que no se vean afectadas cuando cambie la configuración.

Control de transacciones. El depósito gestiona las interacciones entre múltiples partes de tal modo que se mantiene la integridad de los datos cuando existen usuarios concurrentes y en caso de fallo del sistema. Esto suele implicar un bloqueo de registros, admisiones de dos fases, registros de transacciones y procedimientos de recuperación.

Seguridad. El depósito proporciona mecanismos para controlar quién puede visualizar y modificar la información contenida en él. Como mínimo, el depósito debería de imponer contraseñas multinivel y niveles de permiso que fueran asignados a usuarios individuales. El depósito debería de proporcionar también asistencia para copias de seguridad y restauraciones automáticas y también para archivar grupos seleccionados de informaciones, por ejemplo, por proyectos o por aplicaciones.

Consultas e informes de datos ad hoc. El depósito permite acceder directamente a su contenido mediante una interfaz de usuario cómoda tal como SQL, o mediante un browser

orientado a formularios, haciendo posible un análisis definido por el usuario que va más allá de los informes estándar proporcionados por el conjunto de herramientas CASE.

Apertura. Los depósitos suelen proporcionar un mecanismo de importación/exportación sencillo que hace posible las cargas o transferencias de información al por mayor. Las interfaces suelen ser una transferencia de archivos ASCII plana o una interfaz estándar SQL. Algunos depósitos poseen interfaces de nivel superior que reflejan la estructura de sus metamodelos.

Apoyo multiusuario. Un depósito robusto debe de permitir que múltiples desarrolladores trabajen en una aplicación al mismo tiempo. Debe de gestionar el acceso concurrente a la base de datos mediante múltiples herramientas y por parte de múltiples usuarios, con arbitraje de accesos y bloqueos en el nivel de archivos o registros. Para los entornos basados en redes, el apoyo multiusuario implica también que el depósito dispondrá de una interfaz con protocolos y capacidades de red comunes.

El entorno CASE también efectúa demandas especiales con respecto al depósito que van más allá de lo que está disponible directamente en un SGBD comercial. Las características especiales de los depósitos CASE incluyen:

Almacenamientos de estructuras de datos sofisticadas. El depósito debe admitir tipos de datos complejos tales como diagramas, documentos y archivos, así como sencillos elementos de datos. Un depósito también incluye un modelo de información (o metamodelo) que describe la estructura, relaciones y semántica de los datos almacenados en él. El metamodelo debe de poder extenderse para dar cabida a nuevas representaciones y a informaciones organizativas únicas. El depósito no solamente almacena modelos y descripciones de los sistemas en desarrollo, sino también los metadatos asociados (esto es, una información adicional que describe la información de ingeniería del software en sí, tal como el momento en que se ha creado un componente de diseño concreto, su estado actual y la lista de componentes de los cuales depende).

Imposición de una integridad. El modelo de información del depósito contiene también reglas, o políticas, que describen reglas de negocios válidas y otras restricciones y requisitos acerca de la información que se inserta en el depósito (directamente o a través de una herramienta CASE). Se puede emplear una posibilidad denominada *activador* para activar las reglas asociadas a un objeto siempre que este sea modificado, lo cual hace posible verificar la validez de los modelos de diseño en tiempo real.

Interfaz de herramientas ricas en términos semánticos. El modelo de información del depósito (el metamodelo) contiene una semántica que hace posible que toda una gama de herramientas interpreten el significado de los datos almacenados en el depósito. Por ejemplo, un diagrama de flujo de datos creado mediante una herramienta CASE se almacena en el depósito en un formulario basado en el modelo de información e independiente de toda representación interna que pueda utilizar la herramienta en sí. Entonces otra herramienta CASE puede interpretar el contenido del depósito y emplear la información según la necesite para su tarea. De este modo, la semántica almacenada en un depósito permite compartir datos entre una gama de herramientas, por oposición a las conversiones específicas entre herramientas o entre "puentes".

Gestión de procesos/proyectos. Un depósito contiene información no sólo acerca de la aplicación de software en sí, sino también acerca de las características de cada proyecto particular y del proceso general de la organización para ingeniería del software (fases, tareas y productos). Esto abre posibilidades para la coordinación automatizada de la actividad de desarrollo técnico con la actividad de gestión del proyecto. Por ejemplo, la actualización del estado de las tareas de un proyecto se podría efectuar automáticamente o bien como resultado de la utilización de herramientas CASE. La actualización de estado se podría hacer muy fácil para los desarrolladores, sin tener que abandonar el entorno de desarrollo normal. La asignación de tareas y consultas también se puede gestionar por correo electrónico. Los informes de problemas, las tareas de mantenimiento, las actualizaciones de cambios y los estados de reparación se pueden coordinar y monitorizar mediante herramientas que acceden al depósito.

Las siguientes características del depósito son abarcadas todas ellas por la gestión de configuración del software.

Versiones. A medida que progresa un proyecto, se irán creando muchas versiones de productos individuales. El depósito debe ser capaz de guardar todas estas versiones para hacer posible una gestión efectiva de las versiones de los productos y para permitir que los desarrolladores vuelvan a las versiones anteriores durante la comprobación y depuración. Las versiones se efectúan con un algoritmo de compresión para minimizar la reserva de espacio y permiten la regeneración de cualquier versión anterior con un cierto coste de procesamiento.

Seguimiento de dependencias y gestión de cambios. El depósito gestiona una amplia variedad de relaciones entre los elementos de datos almacenados en él. Entre estas se cuentan las relaciones entre entidades y procesos de la empresa, entre las partes de un diseño de aplicación, entre componentes del diseño y la arquitectura de la información de la empresa, entre elementos de diseño y productos, etc. Algunas de las relaciones son meramente asociaciones y algunas son dependencias o relaciones obligatorias. El mantenimiento de esas relaciones entre objetos de desarrollo se denomina *administración de enlaces.*

La capacidad de seguir la pista de todas estas relaciones es crucial para la integridad de la información almacenada en el depósito y para la generación de productos basados en ella y es una de las contribuciones más importantes que efectúa el concepto de depósito para la mejora del proceso de desarrollo del software. Entre las muchas funciones que apoya la gestión de enlaces se cuenta la capacidad de identificar y estimar los efectos del cambio. A medida que los diseños evolucionan para satisfacer nuevos requisitos, la capacidad de identificar todos los objetos que puedan verse afectados hace posible una estimación más precisa del coste, del tiempo no operativo y del grado de dificultad. También se ayuda a evitar efectos colaterales inesperados que en caso contrario darían lugar a defectos y a fallos del sistema.

Seguimiento de requisitos. Una función especial que depende de la gestión de enlaces es el *seguimiento de requisitos.* Esta es la capacidad de seguir la pista de todos los componentes de diseño y de todos los productos que resulten de una especificación de requisitos concreta (seguimiento progresivo) así como la capacidad de identificar el requisito que haya generado cualquier producto dado (seguimiento regresivo).

Gestión de configuración. Otra función que depende de la gestión de enlaces es la gestión de configuraciones. La capacidad de gestión de configuraciones está íntimamente relacionada con la gestión de enlaces y con las capacidades de seguimiento de versiones para seguir la pista de una serie de configuraciones que representarán hitos específicos del proyecto o de versiones de producción. La gestión de versiones proporciona las versiones necesarias y la gestión de enlaces sigue la pista de las interdependencias. Por ejemplo, la gestión de configuraciones suele proporcionar una característica de *construcción* para automatizar el proceso de transformación de componentes de diseño en productos ejecutables.

Seguimiento de una auditoría. Relacionada con la gestión de cambio está la necesidad de un seguimiento de auditoría que establezca informaciones adicionales acerca de cuándo, por qué y por quién son efectuados los cambios. En realidad, no es un requisito difícil para un depósito que posea un modelo de información robusto. La información acerca de las modificaciones efectuadas en el código fuente se pueden introducir en forma de atributos de objetos específicos del depósito. Un mecanismo de activación en el depósito resultará útil para solicitar al desarrollador o a la herramienta que esté utilizando que inicie la introducción de información de auditoría (tal como la razón del cambio) siempre que se modifique en elemento de diseño.

2.7. INGENIERIA INVERSA

La ingeniería inversa del software es el proceso consistente en analizar un programa en un esfuerzo por crear una representación del programa con un nivel de abstracción más

elevado que el código fuente. La ingeniería inversa es un proceso de recuperación de diseño. Las herramientas de ingeniería inversa extraen información acerca de los datos, arquitectura y diseño de procedimientos de un programa ya existente.

La ingeniería inversa puede extraer la información de diseño aparte del código fuente, pero el nivel de abstracción, la completitud de la documentación, el grado con el cual trabajan al mismo tiempo las herramientas y el analista humano, y la direccionalidad del proceso son sumamente variables.

El *nivel de abstracción* de un proceso de ingeniería inversa y las herramientas que se utilicen para realizarlo aluden a la sofisticación de la información de diseño que se puede extraer del código fuente. Idealmente, el nivel de abstracción sería lo más alto posible. Esto es, el proceso de ingeniería inversa debería de ser capaz de derivar sus representaciones de diseño de procedimientos (con un bajo nivel de abstracción); y la información de programas de estructuras de datos (un nivel de abstracción ligeramente más elevado); modelos de flujo de datos y de control (un nivel de abstracción relativamente alto); y modelos de entidades y de relaciones (un elevado nivel de abstracción). A medida que crece el nivel de abstracción se proporciona al ingeniero del software información que le permitirá una comprensión más sencilla de estos programas.

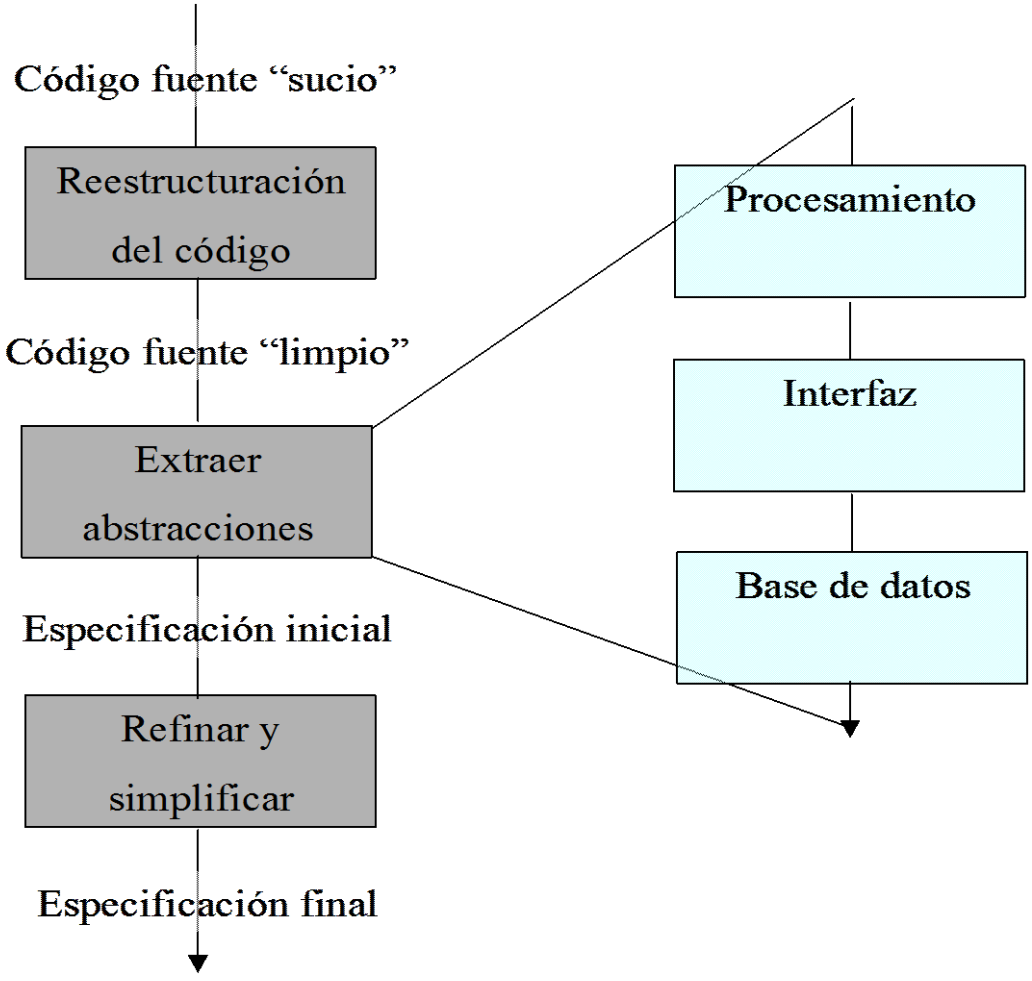
La *completitud* de un proceso de ingeniería inversa alude al nivel de detalle que se proporciona en un determinado nivel de abstracción. En la mayoría de los casos, la completitud decrece a medida que aumenta el nivel de abstracción. Por ejemplo, dado un listado del código fuente, es relativamente sencillo desarrollar una representación de diseño de procedimientos completa. También se pueden derivar sencillas representaciones del flujo de datos, pero es mucho más difícil desarrollar un conjunto completo de diagramas de flujo de datos o un diagrama de transición de estados.

La completitud mejora en proporción directa con la cantidad de análisis efectuado por la persona que efectúe la ingeniería inversa. La interactividad alude al grado con el cual el ser

humano se “integra” con unas herramientas automatizadas para crear un proceso de ingeniería inversa efectivo. En la mayoría de los casos, a medida que crece el nivel de abstracción, la interactividad debe de incrementarse o bien la completitud se verá reducida.

Si la *direccionalidad* del proceso de ingeniería inversa es monodireccional, toda la información extraída del código fuente se proporcionará a la ingeniería del software que podrá entonces utilizarla durante la actividad de mantenimiento. Si la direccionalidad es bidireccional, entonces la información que se suministrará a una herramienta de la reingeniería que intentará reestructurar o regenerar el viejo programa.

Figura. 2.4



El proceso de la ingeniería inversa se representa en la figura 2.4. Antes de que pueda comenzar las actividades de ingeniería inversa, el código fuente no estructurado (“sucio”) se reestructura para que solamente contenga estructuras de programación estructurada. Esto hace que el código fuente sea más fácil de leer y proporciona la base para todas las actividades de la ingeniería inversa subsiguientes.

El núcleo de la ingeniería inversa es una actividad denominada *extracción de abstracciones.* El ingeniero tiene que evaluar el viejo programa y a partir del código fuente (que suele no estar documentado) tiene que extraer una especificación significativa del procesamiento que se realiza, la interfaz de usuario que se aplica, y las estructuras de datos de programa o de la base de datos que se utiliza.

2.7.1. Ingeniería inversa para comprender el procesamiento.

La primera actividad real de la ingeniería inversa comienza con un intento de comprender y después extraer abstracciones de procedimientos representadas por el código fuente. Para comprender las abstracciones de procedimientos, se analiza el código en distintos niveles de abstracción, sistema, programa, módulo, trama y sentencia.

La funcionalidad general de todo el sistema debe de ser algo perfectamente comprendido antes de que se produzca un trabajo de ingeniería inversa más detallado. Así se establece un contexto para su posterior análisis, y se proporcionan ideas generales acerca de los problemas de interoperabilidad entre aplicaciones dentro del sistema. Cada uno de los programas de que consta el sistema de aplicaciones representará una abstracción funcional con un elevado nivel de detalle. Un diagrama de bloques, que represente la iteración entre estas abstracciones funcionales, se creará también. Los módulos efectúan cada uno de ellos una subfunción, y representan una abstracción de procesos definida. Se crean alternativas de procesamiento para cada uno de los modelos. En algunos casos, ya existen especificaciones de sistema, programa y módulo. Cuando tal cosa ocurre, se revisan las especificaciones para apreciar si se ajustan al código existente.

2.7.2. Ingeniería inversa para comprender los datos.

La ingeniería inversa de datos suele producirse en distintos niveles de abstracción. En el nivel de programa, es frecuente que sea preciso realizar una ingeniería inversa de las estructuras de datos de programa internas, como parte de un esfuerzo global de reingeniería. En el nivel del sistema, es frecuente que se efectúe una reingeniería de las estructuras globales de datos (por ejemplo; archivos, bases de datos) para ajustarlas a los nuevos paradigmas de gestión de bases de datos (por ejemplo; el paso de unos archivos planos a unos sistemas de bases de datos relacionales u orientados a objetos). La ingeniería inversa de las estructuras de datos globales actuales establecen el escenario para la introducción de una nueva base de datos que abarque todo el sistema.

Estructuras de datos internas. Las técnicas de ingeniería inversa para datos de programa internos se centran en la definición de clases de objetos. Esto se logra examinando el código del programa con la intención de agrupar variables de programa que estén relacionadas. En muchos casos, la organización de datos en el seno del código identifica los tipos abstractos de datos. Por ejemplo, las estructuras de registros, los archivos, las listas y otras estructuras de datos suelen proporcionar una indicación inicial de las clases.

Algunos autores sugieren el enfoque siguiente para la determinación de clases para la ingeniería inversa:

1. Se identifican los indicadores y estructuras de datos locales dentro del programa que registren información importante acerca de las estructuras de datos globales (por ejemplo; archivos o bases de datos).
2. Se define la relación entre indicadores y estructuras de datos locales y las estructuras de datos globales. Por ejemplo, se puede activar un indicador cuando está vacío un archivo, o bien una estructura de datos local que puede servir como tapón que contenga los cien últimos registros capturados de una base de datos central.
3. Para toda variable (perteneciente al programa) que represente una matriz o archivo, se enumeran todas las demás variables que tengan una relación lógica con ella.

Estos pasos capacitan al ingeniero de software para identificar las clases dentro del programa que interactúan con las estructuras de datos globales.

Estructuras de base de datos. Independientemente de su organización lógica y de su estructura física, las bases de datos permiten definir objetos de datos, y admiten algún método para establecer relaciones entre objetos. Consiguientemente, la reingeniería de un esquema de bases de datos para formar otro exige una comprensión de los objetos ya existentes y de sus relaciones.

Se pueden emplear los pasos siguientes para definir el modelo de datos existente como precursor para una reingeniería que producirá un nuevo modelo de base de datos:

1. ***Se construye un modelo de objetos inicial***. Las clases definidas como parte del modelo se pueden conseguir mediante la revisión de registros de una base de datos de archivos planos o de tablas de un esquema relacional. Los objetos contenidos en los registros o tablas pasarán a ser atributos de clases.
2. ***Se determinan los candidatos a claves***. Se examinan los atributos para determinar si se utilizarán o no para señalar a otro registro o tabla. Aquellos que sirvan como punteros pasarán a ser candidatos a claves.
3. *Se refinan las clases tentativas.* Se determina si ciertas clases similares pueden o no combinarse en una única clase.
4. *Se definen las generalizaciones.* Se examinan las clases que puedan tener muchos atributos similares para determinar si se debería o no construir una jerarquía de clases con una clase de generalización como precursor de todos sus descendientes.
5. *Se descubren las asociaciones.* Mediante el uso de técnicas análogas al enfoque de CRC (modelado clases – responsabilidades – colaboraciones) se establecen las asociaciones entre clases.

Una vez que se conoce la información definida en los pasos anteriores, se pueden aplicar una serie de transformaciones para hacer corresponder la estructura de la vieja base de datos con una nueva estructura de base de datos.

2.7.3. Ingeniería inversa de interfaces de usuario.

A medida que las IGU (Interfaz Gráfica de Usuario) sofisticadas se van volviendo de rigor para los productos basados en computadoras y para los sistemas de todo tipo, el nuevo desarrollo de interfaces de usuario ha pasado a ser uno de los tipos más comunes de actividades de reingeniería. Ahora bien, antes de que sea posible reconstruir una interfaz de usuario, suele ser necesario que se produzca una actividad de ingeniería inversa.

Para comprender en su totalidad una interfaz de usuario ya existente (IU), es preciso especificar la estructura y comportamiento de la interfaz. Se sugieren tres preguntas básicas a las cuales hay que responder cuando comienza la ingeniería inversa de la IU.

- ¿Cuáles son las acciones básicas que debe de procesar la interfaz, por ejemplo, acciones de teclado y clics de ratón?
- ¿Cuál es la descripción compacta de la respuesta del sistema a estas acciones?
- ¿Qué quiere decir una "sustitución", o más exactamente, qué concepto de equivalencia de interfaces es relevante en este caso?

La descripción de la interfaz hace uso de agentes y acciones. Un agente es algo que da vida a algún aspecto del sistema. Las acciones permiten que los agentes se comuniquen entre sí.

2.8. HERRAMIENTAS CASE ACTUALES

2.8.1. DESIGNER /2000

¿Qué es Oracle Designer/2000?

Oracle Designer/2000 es una herramienta de modelado para el desarrollo de aplicaciones en arquitectura cliente/servidor, incorporando apoyo para el análisis de sistemas, diseño de software, generación de código automática y reingeniería de procesos de negocio.

Componentes de Designer/2000:

Business Process Modeling:

- Process Modeller (BPR)

Systems Analysis Modeling:

- Entity Relationship (ERD)
- Function Heirarchy (FHD)
- Dataflow Diagrammer (DFD)

Design Wizards:

- DB Design Wizard (DDW)
- Application Design Wizard (ADW)

Systems Design:

- Data Diagrarmmer (DD)
- Module Logic (MLD)
- Module Data (MDD)
- Preference Navigator (PN)
- Module Structure (MSD)

Client/Server Generators:

- Server Generator
- Graphics Generator
- Web Server Generator
- Forms Generator (CGENF45)
- Reports Generator (CGENR25)
- Visual Basic Generator (VBGEN10)
- C++ Object layer Generator (CPPGEN10)
- MSHelp Generator

Utilities:

- Matrix Diagrammer (MD)
- Repository Object Navigator (RON)

- Repository Administrator (RAU)
- Repository Reports (REP)
- Repository Utilities (UTL)

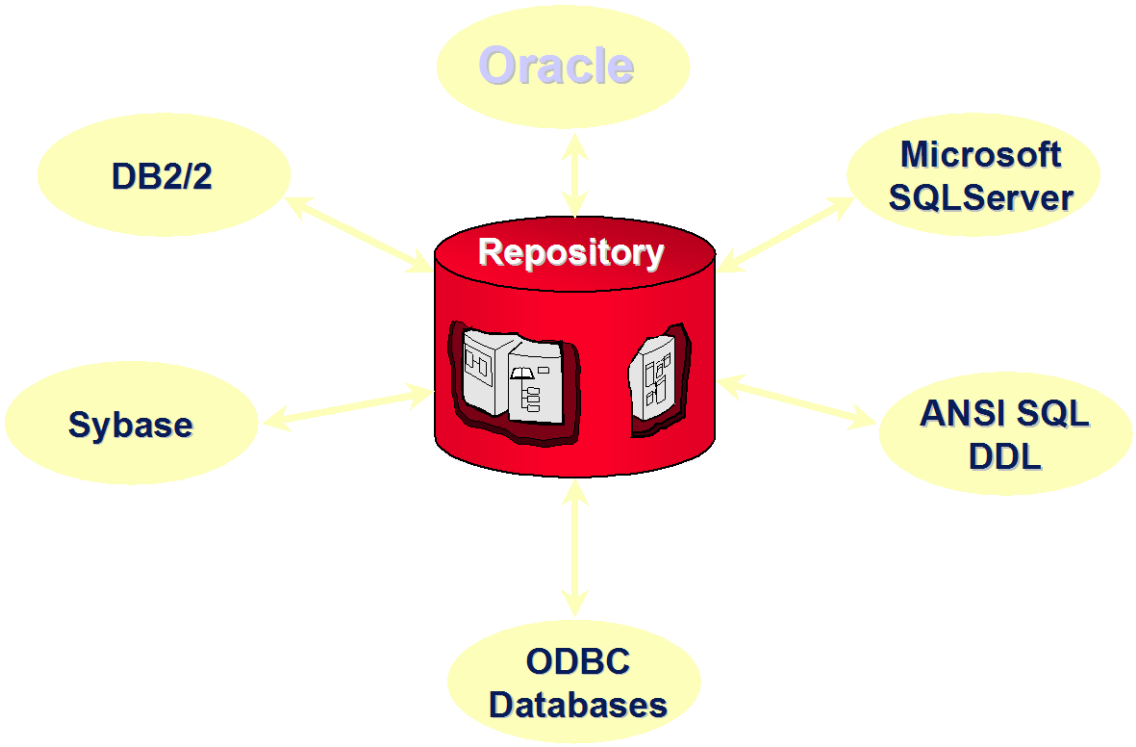
Limitaciones

Una de las limitantes de esta herramienta es la utilización de muchos recursos del computador, tanto en almacenamiento como en memoria (mínimo 64 Mb).

Características

- Puede generar bases de datos para la mayoría de manejadores de bases de datos conocidos en el mercado (Fig. 2.5).

Figura. 2.5



- Permite trabajar en equipos de trabajo en una sola aplicación.
- Mantiene un control de cambios sobre proyectos que se estén diseñando.

- Existe un nivel de control para los usuarios, por lo tanto existe control para proyectos que ellos desarrollan.
- Existen herramientas para modelar procesos de negocios.

2.8.2. PowerDesigner (Sybase)

Descripción.

El sistema relacional de base de datos fue originalmente diseñado para soportar procesamiento de transacciones en línea y consultas a la base de datos. Sin embargo, el modelo relacional está limitado en términos de soportar tipos de datos definidos por el usuario y acceso a la información de la base de datos por aplicaciones de negocio orientadas a objetos o componentes.

En la versión 7.0, PowerDesigner se ha reforzado para soportar diseños de base de datos objeto - relación. Además de automatizar el diseño, mantenimiento y recuperación de las aplicaciones de la base de datos relacional, PowerDesigner 7.0 ayuda a diseñar tipos de datos definidos por el usuario para ser almacenados en la base, así como la lógica del negocio usada para accesar y manipular la información de la base de datos.

PowerDesigner 7.0 reúne las necesidades de diseñadores de base de datos y desarrolladores de aplicación otorgando la funcionalidad requerida por ambos:

- Diseño y generación de esquemas de base de datos (soporte para más de 30 sistemas manejadores de base de datos).
- Diseño y generación de la lógica del negocio orientada a objetos (Java y PowerBuilder de Sybase).

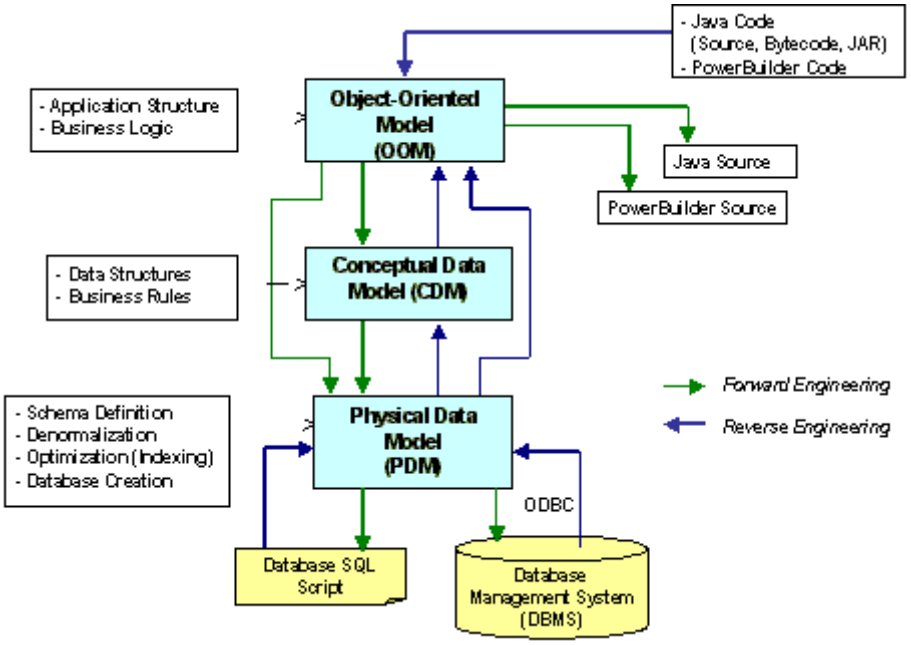
PowerDesigner 7.0 extiende su tradicional nivel - dual, ambiente gráfico de modelado de datos con un análisis orientado al objeto y diseño (OOA&D). Con PowerDesigner 7.0, las definiciones de base de datos y componentes del negocio orientado al objeto están

transparentemente relacionados e integrados en un ambiente homogéneo, usuario - amigable, y un ambiente de herramientas de diseño muy productivo.

Como muestra la figura abajo (Ciclo de vida del diseño de una aplicación con PowerDesigner7) PowerDesigner ofrece una opción al usuario del paradigma del modelado:

- **El modelado orientado al objeto**: La definición de una estructura de la aplicación en un Lenguaje de Modelado Unificado (UML, Unified Modeling Language) de diagramas de clase, y definición de la lógica del negocio que opera en estructuras de la clase. Desde un diagrama de la clase, PowerDesigner permite generar desde el código fuente de Java y PowerBuilder.
- **Modelado conceptual de Datos:** La definición de estructuras de datos abstractas en diagramas Entidad Relación (ER), junto con reglas del negocio basadas en requisitos del usuario. Un modelo conceptual de datos puede generarse en un modelo de datos físico. Recíprocamente, un modelo de datos físico puede usar ingeniería - reversa para obtener un modelo conceptual.
- **Modelado físico de datos:** La definición del esquema de la base de datos, denormalización, optimización de consultas, y la creación de la base de datos física (generación de scripts del lenguaje de definición de datos DDL) o la creación de la base de datos real a través de una conexión de ODBC.)

Fig. 2.6



Interface de Usuario.

Características	Descripción / Beneficios
Arquitectura del producto basada en objetos.	• Cada parte de información manipulada por PowerDesigner es un "objeto" identificado y desplegado en el navegador de objetos de PowerDesigner. • Información (objeto) es reutilizada por las aplicaciones.
Núcleo común.	• Toda la funcionalidad de PowerDesigner es ahora accesible desde una interface del usuario unificada organizada en una ventana primaria. • La Información es transparentemente compartida y sincronizada entre los tipos de modelos (Modelos orientados a objetos, modelos conceptuales de datos, modelos físicos de datos.)
Ambiente personalizado.	• Las barras de herramientas son movibles y pueden ser personalizadas individualmente usando iconos. • Despliegue de preferencias (cómo cada objeto se ve y cómo está estructurado) • Opciones ejemplares (nombrando convenciones específicas para cada diseño de objeto usando nombres de plantillas de PowerDesigner)

Navegador de objetos.	• La estructura de árbol que incluye todos los objetos que son parte del espacio de trabajo, como son carpetas, modelos, paquetes y más información bien formada como son clases, atributos, tablas, índices, referencias, disparadores, etc. • Puede arrastrar y colocar documentos que no son de PowerDesigner como son hojas de cálculo y documentos de procesadores de palabras desde el explorador de Windows al navegador de objetos de PowerDesigner, y abrirlos directamente desde el navegador.
Personalizable, reestructurable, modelos hoja – propiedad, lista - objetos, ventanas cuadro - diálogo.	• Puede abrir varias ventanas para un solo objeto simultáneamente. • Puede cortar y pegar o arrastrar y colocar información entre varias ventanas. • Puede seleccionar (filtrar) la información que usted quiere desplegar en una ventana de hoja de propiedad. • Puede grabar la personalización, tamaño, y posición en pantalla de cada ventana.
Modos Standard / Auto-Commit	• El modo standard de validación permite crear y modificar objetos en una hoja de propiedad, y valida información al cerrar la hoja de propiedad. • El modo Auto-Commit de validación permite realizar (grabar a memoria) cualquier modificación hecha en una hoja de propiedad en cuanto sea ingresado.

2.9. EVALUACION DE HERRAMIENTAS.

Las herramientas anteriormente descritas presentan sus ventajas una con respecto a la otra, es así que:

- PowerDesigner de Sybase es una herramienta que está trabajando con el enfoque objeto – relación, en cambio Designer/2000 de Oracle utiliza el modelado entidad – relación.
- Hay que considerar que Oracle es uno de los mejores manejadores de bases de datos y esto beneficia enormemente en la difusión de su herramienta CASE.
- De acuerdo a sus requerimientos PowerDesigner no es un gran consumidor de recursos como lo es Designer/2000, pero en empresas donde se maneja gran cantidad de

información se debe disponer de grandes recursos en hardware y se justificaría el uso de la herramienta CASE de Oracle.

- Ambas herramientas comparten características comunes, tienen modeladores de procesos de negocio, facilidades para generar y capturar diseños físicos de otras bases de datos (SQL Server, DB2, etc.)

Para este proyecto se utilizó Designer/2000 para construir el respectivo diagrama Entidad – Relación, puesto que EMELNORTE cuenta actualmente con el manejador de bases de datos Oracle y esta herramienta estaba incluida, la conexión con la base de datos no presentaría ningún problema puesto que son productos de la misma empresa.