



**INSTITUTO POLITÉCNICO NACIONAL**

---

**CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN**

**SISTEMA BASADO EN CONOCIMIENTO COMO  
HERRAMIENTA DE APOYO PARA LA ADMINISTRACIÓN  
DE PROYECTOS DE INGENIERÍA DE *SOFTWARE* - HIS**

**T E S I S**

QUE PARA OBTENER EL GRADO DE MAESTRO EN

**CIENCIAS DE LA COMPUTACIÓN**

**P R E S E N T A:**

**FELIPE JUÁREZ RODRÍGUEZ**

**DIRECTOR:  
CODIRECTOR:**

**DR. BÁRBARO JORGE FERRO CASTRO  
DR. ÁLVARO DE ALBORNOZ BUENO**

**MÉXICO D.F.**

**MARZO 2003**



**Agradezco al**

**CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN  
del  
INSTITUTO POLITÉCNICO NACIONAL**

**y al**

**INSTITUTO MEXICANO DEL PETRÓLEO**

**Por permitir realizarme como mejor profesionista y persona**

**Y muy especialmente a los profesores asesores que me apoyaron en la  
elaboración de este trabajo de Tesis**

**Dr. Bárbaro Jorge Ferro Castro**

**Dr. Álvaro de Albornoz Bueno**

**Dra. Patricia Rayón Villela**

**Dr. Aurelio Velázquez**

**Dr. Sergio Suárez Guerra**

**Dra. Olivia Niño Leal**

**Así como a todas las personas que me ayudaron de diferentes formas para  
lograr este objetivo**

**A mi MADRE:**

**Por sus consejos y ejemplos de vida**

**A mis hijos:**

**Felipe, Diana, Gabriela y Luis F.**

**A la memoria de mis familiares  
ausentes y a Rodrigo de Jesús  
por su valentía**

**A mi esposa LETY:**

**Por su apoyo, comprensión y compañerismo**

**A IRENE:**

**Con respeto**

# INDICE

<b>RESUMEN</b>	V
<b>ABSTRACT</b>	VII
<b>GLOSARIO DE TERMINOS</b>	IX
<b>RELACIÓN DE FIGURAS</b>	XII
<b>CONTENIDO</b>	
<b>Capítulo I.- Introducción</b>	1
I.1.- Problemática detectada en el desarrollo de Proyectos de Ingeniería de <i>Software</i>	2
I.2.- Metodología de desarrollo de la Solución	4
I.3.- Objetivos y Alcances	6
I.4.- Beneficios y Contribuciones	9
I.5.- Motivación y Justificación de desarrollo	10
I.6.- Resultados esperados	12
I.7.- Descripción del contenido	13
<b>Capítulo II.- Herramientas existentes de apoyo a la Ingeniería de <i>Software</i> “Estado del Arte”</b>	15
II.1.- <i>Software</i> para Administración de Proyectos	15
II.2.- Herramientas CASE	17
II.2.1.- CASE para diseño gráfico por computadora +1 <i>Software Engineering</i>	19
II.2.2.- Designer 2000 de Oracle	19
II.2.3.- CASE Rational Rose	20
II.3.- Sistemas Inteligentes para la Ingeniería de <i>Software</i>	20
II.3.1.- Perspectivas en la automatización de las Metodologías de Ingeniería de <i>Software</i>	20
II.3.2.- MeRCI Sistema Experto para Reingeniería de <i>Software</i>	21
II.3.3.- Herramienta ICASE para diseño reusando conocimientos adquiridos	22
II.3.4.- Comparación de Sistemas Inteligentes para la Ingeniería de <i>Software</i>	24
<b>Capítulo III.- Dominio del Conocimiento de la HIS para el PROTOTIPO SE-APIS “Ingeniería de <i>Software</i>”</b>	26
III.1.- Módulo de Conocimiento de Metodologías de Desarrollo	27
III.1.1.- Modelos del ciclo de vida de un sistema	27
III.1.2.- Ciclo de vida clásico	31
III.1.3.- Construcción de prototipos	32
III.1.4.- Modelo Orientado a Objetos	33
III.2.- Módulo de Conocimiento de Administración de Proyectos de <i>Software</i>	35

III.2.1.- Fases y ciclo de vida de un proyecto de <i>software</i>	35
III.2.2.- WBS, definición del alcance de un proyecto de <i>software</i>	37
III.2.3.- Planeación del tiempo del proyecto de <i>software</i>	39
III.3.- Módulo de Conocimiento de Métricas de Desarrollo	44
III.3.1- Modelo de Productividad de <i>Software</i> de PUTNAM	44
<b>Capítulo IV.- Fuentes de Conocimiento y Biblioteca de CASOS</b>	<b>46</b>
IV.1.- Identificación de las fuentes de conocimiento, FCP y FCA	46
IV.2.- Elicitación del Conocimiento	47
IV.3.- Biblioteca de CASOS de Metodologías de Desarrollo	48
<b>Capítulo V.- Técnicas de la Inteligencia Artificial para el desarrollo del SE-APIS</b>	<b>50</b>
V.1.- Representación del conocimiento	50
V.1.1.- Aproximaciones a la representación del conocimiento	50
V.1.2.- Problemas de la representación del conocimiento	54
V.1.3.- Selección de la granularidad de la representación	56
V.2.- Estructuras de ranura y relleno	57
V.2.1.- Estructura de ranura y relleno débiles: <i>Marcos</i>	57
V.2.2.- Estructura de ranura y relleno fuertes: Dependencia Conceptual	59
V.3.- Procesamiento de Lenguaje Natural	63
V.3.1.- Análisis morfológico	64
V.3.2.- Procesamiento sintáctico	64
V3.3.- Análisis semántico	67
<b>Capítulo VI.- Diseño del PROTOTIPO SE-APIS del Sistema Basado en Conocimiento</b>	<b>70</b>
VI.1.- Especificaciones de procesamiento	70
VI.2.- Modelo de <i>Marcos</i> de conocimiento del SE-APIS	72
VI.2.1.- Modelo de las Metodologías de Ingeniería de <i>Software</i>	72
VI.2.2.- Modelo de Administración de Proyectos	74
VI.3.- Modelo de procesamiento del SE-APIS	79
VI.3.1.- Procesamiento de Lenguaje Natural del SE-APIS	80
VI.3.2.- Proceso de Proyectos de Ingeniería de <i>Software</i> del SE-APIS	86
VI.3.3.- Aplicación de la Métrica del Modelo de Putnam en los Proyectos	90
VI.3.4.- Calculo de la ruta crítica de los proyectos del SE-APIS	91
VI.4.- Modelo de proceso del PROTOTIPO SE-APIS	92
<b>Capítulo VII.- Construcción del PROTOTIPO SE-APIS</b>	<b>94</b>
VII.1.- Construcción en CLIPS	94
VII.2.- Módulos componentes del SE-APIS	94
VII.3.- Construcción del módulo de la Base de Conocimientos de <i>Marcos</i> , Metodos2.clp	96
VII.4.- Representación de las Instancias y Hechos de Conocimiento, Instancias.clp	98
VII.5.- Construcción de la Interfaz de Procesamiento de Lenguaje Natural, Interface1.clp	101

VII.6.-	Generación de Proyectos de Ingeniería de <i>Software</i> y Métrica de Putnam, Projects.clp	104
VII.7.-	Cálculo de la Ruta Crítica de los Proyectos, Ruta_Crítica1.clip	107
VII.8.-	Respuestas y presentación de resultados para el usuario, Answer.clp	110
VII.9.-	Métricas, número de reglas de inferencia y líneas de código del SE-APIS	112
<b>Capítulo VIII.- Evaluación de resultados del PROTOTIPO SE-APIS</b>		<b>113</b>
VIII.1.-	Diseño de pruebas del SE-APIS	113
VIII.2.-	Ejecución del SE-APIS en CLIPS	115
VIII.3.-	Escenario de Evaluación de la Interfaz de Lenguaje Natural	116
VIII.4.-	Evaluación de Consulta y reportes de Procedimientos y Metodologías	120
VIII.5.-	Escenario de Evaluación de Generación de Proyectos y Métricas de Putnam	127
VIII.6.-	Escenario de Evaluación de Cálculo de la ruta crítica de Proyectos	130
<b>Capítulo IX.- Trabajos futuros de la HIS</b>		<b>133</b>
IX.1.-	Modelo Final de un Sistema Experto Integral HIS	133
IX.2.-	Incorporación de métricas de desarrollo de <i>Software</i>	135
IX.3.-	Desarrollo en ambiente Internet	136
<b>CONCLUSIONES</b>		<b>137</b>
<b>REFERENCIAS BIBLIOGRÁFICAS</b>		<b>139</b>
<b>ANEXOS (Disco compacto)</b>		
A.-	Código de programación del SE-APIS	
B.-	Método de desarrollo de aplicaciones CASE (CADM)	



## RELACIÓN DE FIGURAS

Figura I.1	Problemas detectados en proyectos de Ingeniería de <i>Software</i>	3
Figura I.2	Metodología de desarrollo del Sistema Experto como apoyo a la Administración de proyectos de ingeniería de <i>Software</i>	5
Figura I.3	Modelo de un Sistema Inteligente Integral y del PROTOTIPO SE-APIS	8
Figura II.1	Comparación de características de <i>software</i> para Administración de proyectos	16
Figura II.2	Un banco de trabajo para el análisis y diseño	18
Figura II.3	Modelo de SE con metodologías E-R-E y DOO	20
Figura II.4	Estructura de una herramienta CASE inteligente	23
Figura II.5	Cuadro comparativo de Sistemas Inteligentes de Ingeniería de <i>Software</i>	25
Figura III.1	Modelos secuenciales de cascada	28
Figura III.2	Modelo iterativo de cascada	28
Figura III.3	Modelo “b”	29
Figura III.4	Modelo de espiral	30
Figura III.5	El ciclo de vida clásico	31
Figura III.6	Construcción de prototipos	32
Figura III.7	Ciclo de vida orientada a objetos	34
Figura III.8	Ciclo de vida típico de un proyecto de <i>software</i>	36
Figura III.9	Ciclo típico de un proyecto de <i>software</i>	37
Figura III.10	Work Breakdown Structure de un proyecto de <i>software</i>	38
Figura III.11	Diagrama del método de modelo de precedencias (PDM)	40
Figura III.12	Relación entre actividades	41
Figura III.13	Gráfica de precedencias con una actividad <i>dummy</i>	41
Figura III.14	Red (grafo) de actividades	42
Figura III.15	Cálculo de duración PERT	44
Figura III.16	Modelo de Putnam estimado de costo y ciclo de <i>software</i>	45
Figura IV.1	Caso específico prototipo CADM	49
Figura V.1	Representación del conocimiento	50
Figura V.2	Conocimiento relacional simple	52
Figura V.3	Conocimiento heredable	52
Figura V.4	Un nodo visto como un <i>marco</i> (frame)	53
Figura V.5	Conocimiento deductivo	53
Figura V.6	Conocimiento procedimental expresado en forma de reglas	54
Figura V.7	Un sistema de <i>marcos</i> simplificado	58
Figura V.8	Una sencilla representación en Dependencia Conceptual	59
Figura V.9a	Las dependencias de la Dependencia Conceptual (CD)	61
Figura V.9b	Las dependencias de la Dependencia Conceptual (CD)	62
Figura V.10	Una sencilla gramática para un subconjunto del español	65
Figura V.11	El árbol de análisis de una frase	66
Figura V.12	Una gramática semántica	68
Figura V.13	El resultado de análisis con una gramática semántica	69
Figura VI.1	Modelo de relaciones entre clases de metodologías de Ingeniería de <i>software</i>	73

## RELACIÓN DE FIGURAS (continúa)

Figura VI.2	Definición y WBS de un proyecto de Ingeniería de <i>software</i>	74
Figura VI.3	Modelo de relaciones entre clases de proyectos de Ingeniería de <i>software</i>	75
Figura VI.4	Modelo completo de relaciones entre clases de Ingeniería de <i>software</i>	77
Figura VI.5	Modelo de procesamiento del sistema experto APIS	79
Figura VI.6	Procesamiento del análisis morfológico del SE-APIS	81
Figura VI.7	Estructura jerárquica del analizador sintáctico del APIS	82
Figura VI.8	Procesamiento del análisis sintáctico del SE-APIS	83
Figura VI.9	Procesamiento de la generación de proyectos, CD crea()	86
Figura VI.10	Generación de las WBS y OPERACIONES del proyecto	87
Figura VI.11	Curvas paramétricas de un proyecto	88
Figura VI.12	Datos estadísticos de diferentes proyectos	88
Figura VI.13	Curva estadística de proyectos de Ingeniería de <i>software</i>	89
Figura VI.14	Pesos porcentuales de las fases de la metodología CADM	89
Figura VI.15	Red de fases WBS de un proyecto de Ingeniería de <i>software</i>	91
Figura VI.16	Proceso de cálculo de la ruta crítica en el SE-APIS	91
Figura VI.17	Modelo general del proceso del PROTOTIPO del SE-APIS	92
Figura VII.1	Diagrama de proceso de los módulos de programación del SE-APIS	95
Figura VII.2	Métricas del SE-APIS	112
Figura VIII.1	Formato para comparación de valores calculados por el SE-APIS	114
Figura VIII.2	Formato para pruebas de validación modular e interfaces del SE-APIS	114
Figura VIII.3	Formato para verificación de defectos de los módulos del SE-APIS	114
Figura VIII.4	Evaluación del procesamiento de Lenguaje Natural	119
Figura VIII.5	Excepciones del procesamiento de Lenguaje Natural	120
Figura VIII.6	Evaluación de consultas y reportes de procedimientos y metodologías	127
Figura VIII.7	Excepciones de consultas y reportes de procedimientos y metodologías	127
Figura VIII.8	Comparación de valores calculados para el SE-APIS	130
Figura VIII.9	Verificación de defectos de los módulo Projects.clp	130
Figura VIII.10	Red para autoevaluación del SE-APIS	131
Figura VIII.11	Excepciones del módulo de Ruta_crítica	132

## RESUMEN

En este trabajo de tesis se presenta un Sistema Basado en Conocimiento como Herramienta de Ingeniería de *Software* **HIS**, que servirá de apoyo a la Administración de Proyectos de Ingeniería de *Software*, y que denominaremos a lo largo del trabajo de tesis con el nombre de **SE-APIS**, para diferenciarlo de una **HIS integral**. El **SE-APIS** permite realizar consultas y reportes acerca de metodologías, etapas y actividades de Ingeniería de *Software*. Asimismo se crean en forma automática las estructuras prototipo de Proyectos para Ingeniería de *Software*, que permiten al equipo de desarrollo de sistemas contar con una herramienta de Administración de Proyectos de *Software* y del ciclo de vida de los sistemas de información.

El SE-APIS es un sistema experto basado en conocimientos (Knowledge Based Expert System), estructurado con **Marcos** (Minsky, 1975 [MIN75]) u objetos (Grady Booch, 1995, [POO98]) que tiene varios niveles de conocimiento relacionados por niveles de clases; iniciando desde el primer nivel con la superclase **Procedimientos**, que contiene una estructura de conocimientos para información general de encabezados de procedimientos de metodologías y de Administración de Proyectos de Ingeniería de *Software*. Esta superclase se divide a su vez en dos ramas de clases de conocimiento denominadas **Metodologías** y **Proyectos**.

La clase **Metodologías** se encuentra ubicada en el segundo nivel contiendo una estructura para conocimiento general de cada tipo de Metodología, y ramificándose en un tercer nivel relacionado con la clase **Fases**, la cual almacena conocimientos de distintas etapas de Metodologías. En el cuarto nivel se encuentra relacionada la clase **Actividades** con conocimiento de los detalles que se realizan dentro de cada etapa de las Metodologías.

De la misma forma, en la otra rama de segundo nivel se encuentra la clase **Proyectos**, en la que se tienen conocimientos sobre definiciones de Administración de Proyectos. En el tercer nivel de esta rama se tiene relacionada la clase **WBS** (*Work Breakdown Structure*). En el cuarto nivel se tiene relacionada la clase **Operaciones** de los proyectos que sirve para determinar la ruta crítica de los Proyectos de *Software* en conjunto con un *Marco* llamado **REDES** del mismo nivel de conocimiento, y en un quinto nivel de esta rama se tiene incorporado en el modelo una clase para **Recursos** con los requerimientos humanos, materiales y servicios para el desarrollo de los Proyectos (PMI Project Management Institute, 1996, [PMI96]).

Existe también dentro del modelo de conocimientos del SE-APIS, en un quinto y sexto nivel las estructuras de las clases **Modelos y Métodos** que están relacionadas de forma dependiente con las ramas de **Metodologías y Proyectos**, y que servirán para almacenar conocimientos como: análisis de requerimientos, métodos de bases de datos relacionales (Korth Silberschatz [FBD93]), modelos contextuales y de flujo de datos (Edward Yourdon [EDY93]), que pueden formar parte de los estándares para el desarrollo de *software* dentro del Instituto Mexicano del Petróleo. Si bien, las estructuras de *Marcos* de los Modelos y Métodos se encuentran definidos en la base de conocimientos, las reglas de inferencia de estos, no están desarrollados dentro del trabajo de esta tesis.

La información del dominio del SE-APIS almacenada en los *Marcos* de la Base de Conocimientos, está basada en conceptos de Ingeniería de *Software* [RPE02] y [RPE90] así

como en la Metodología para desarrollo de aplicaciones CASE-CADM (Computer Aided *Software Engineering* – CASE Application Development Method [DES00]) de la compañía Oracle, a la cual se le han adicionado factores de ponderación, obtenidos estadísticamente para cada una de las etapas y actividades del proyecto de acuerdo con los métodos PMI para la Administración de Proyectos, y que se utilizan para calcular automáticamente el tiempo, recursos en horas y costos requeridos para cada una de las fases de los Proyectos de Ingeniería de *Software*.

Asociadas a las estructuras de *Marcos* se desarrollaron reglas de inferencia para consultar y realizar reportes de la Base de Conocimientos, así como reglas para la generación automática de Proyectos de Ingeniería de *Software* y programación de su ruta crítica. Algunas de las reglas construidas, son para el cálculo de los esfuerzos humanos requeridos para el desarrollo de *software* utilizando el Método de la ecuación de Putnam [PUT78] y [PUT92]. Adicionalmente para la Interfaz con el Usuario se desarrollaron reglas de inferencia para la interpretación de una gramática en Lenguaje Natural. Finalmente para el correcto funcionamiento del SE-APIS se construyeron reglas de control para la activación de los mecanismos de inferencia.

Las técnicas para la construcción del SE-APIS pertenecen al dominio de la Inteligencia Artificial y son entre otra técnicas: estructuras de ranura y relleno débiles ***Marcos*** (Minsky, 1975[MIN75]), estructuras de ranura y relleno fuertes ***Dependencias Conceptuales CD*** (Schank, 1975 [SHA75]) así como gramáticas de contexto libre mediante declarativas de **Lenguaje Natural LN** (Allen 1987 [ALL87] y Grosz 1986 [GRO86] ) las cuales son convertidas en CD de consulta y reportes de Metodologías, y en CD para la generación automática y programación de Proyectos de Ingeniería de *Software*.

La metodología utilizada para el desarrollo del Modelo del SE-APIS recibe el nombre de KAMET (Knowledge Acquisition from Multiple Sources, Cairo, 1998 [CAI98]), con la cual fue posible guiar cada una de las etapas de desarrollo, iniciando desde la detección de la problemática, el diseño de la arquitectura del Modelo, hasta la construcción de un PROTOTIPO operable del SE-APIS y la definición de un Modelo Final para la construcción de un Sistema Experto Integral como Herramienta para la Ingeniería de *Software* (HIS).

La programación del SE-APIS se realizó con el *software* para desarrollo de Sistemas Expertos CLIPS [CLI97] (C Language Integrated Production System) de la NASA/Lyndon B. Jhonson Space Center, aprovechando su facilidad de lenguaje de programación orientado a objetos (COOL) para diseñar *marcos* de conocimiento, su potencialidad para definir hechos y reglas de conocimiento para procesar declarativas en Lenguaje Natural, y su flexibilidad para construir CD que puedan ser fácilmente procesadas con las reglas de inferencia.

El SE-APIS fue evaluado de forma cuantitativa y cualitativa con varias consultas y reportes por medio de oraciones en lenguaje natural limitado así como con la generación automática y cálculo de la ruta crítica de varios Proyectos de Ingeniería de *Software*. Incluso con el SE-APIS se realizó su propia evaluación para las etapas de análisis, diseño y construcción, obteniéndose resultados satisfactorios.

Se deja el planteamiento para desarrollar en trabajos futuros de tesis, un Sistema Experto como Herramienta Integral para la Ingeniería de *Software* (HIS), que debe incluir facilidades de interconexión con otras herramientas de soluciones específicas CASE, *software* de métricas de excelencia de desarrollo, y su operación en ambiente Internet.

## **ABSTRACT**

In this job of thesis an Knowledge Based System is presented as Software Engineering Tool (**HIS**), that support the Projects Administration and we have named **SE-APIS**. The **SE-APIS** permits to carry out consultations and reports about methodologies, phases and Software Engineering activities. Likewise they are created in automatic form the structures prototipo of Projects for Engineering of Software that permit to the systems development team to count on a Software Projects Administration tool and of the cycle of life of the systems of information.

The SE- APIS is an expert system based on knowledge (Knowledge Based Expert System), structured with **Frameworks** (Minsky, 1975 [MIN75]) or objects (Grady Booch, 1995, [POO98]) that has various levels of knowledge related by levels of classes; since the first level with the highest class **Procedures**, that contains a structure of knowledge for general information of headlines of procedures of methodologies and of Software Engineering Projects Administration. This highest class is divided at the same time in two knowledge classes branches called **Methodologies and Projects**.

The **Methodologies** class is found located in the second level containing a structure for general knowledge of each type of Methodology, and branching in a third level related to the **Phases** class, which stores knowledge of distinct phases of methodologies. In the fourth level is found related the **Activities** class with knowledge of the details that are carried out inside each phase of the methodologies.

At the same time, in the other branch of second level the **Projects** class is found, which contains knowledge about Projects Administration definitions. In the third level of this branch is related the **WBS** class (Work Breakdown Structure). In the fourth level has related the **Operations** class of the projects that serves to determine the critical route of the Projects of Software as a group with a Framework called **REDES** of the same level of knowledge, and in a fifth level of this branch has incorporated in the model a class for **Resources** with the needed human hours, material and services for the development of the Projects (**PMI** Project Institute, 1996, [PMI96]).

Inside the model of knowledge of the SE-APIS, in a fifth and sixth level are the structures of the **Models** classes and **Methods**, that are related to dependent form with the classes of **Methodologies** and **Projects**, and that will serve to store knowledge as: analysis of requirements, relationed data bases methods (Korth Silberschatz [FBD93]), context models and data flow (Edward Yourdon [EDY93]), that can form standards for the development of software inside the Mexican Petroleum Institute. As well, the structure of Frameworks of the Models and Methods are defined in the base of knowledge, the rules of inference of these, were not developed in the present job.

The information of the control of the SE-APIS stored in the Frameworks of the Base of Knowledge, is based on Software Engineering concepts [RPE02] and [RPE90] as well as the methodologies for CASE-CADM applications development (Computer Aided Software Engineering – CASE Application Development Method [DES00]) by the company Oracle. Factors of percents have been added to CADM, that were statistically obtained for each one of the phases and activities according to the PMI method for the Project Management, that

are used to calculate automatically the time, resources in hours and costs required for each one of the phases of the Software Engineering Projects.

Rules of inference were developed associates to the structures of Frameworks to consult and carry out reports of the Base of Knowledge; likewise were planned rules for the automatic generation of Software Engineering Projects and programming of their critical route. Some of the rules built, are for the calculation of the human efforts required for the development of software using the method of the equation of Putnam [PUT78] and [PUT92]. Additionally for the user interfase, rules of inference were developoped for the interpretation of a grammar in Natural Language. And finally for the correct operation of the SE-APIS, rules of control were built for the activación of the inference mechanisms.

The techniques for the construction of the SE-APIS belong to the Artificial Intelligence Environment and are among another techniques: slot structures and weak backfill “**Frameworks**” (Minsky, 1975[MIN75]), slot structures and strong backfill “**Conceptuals Dependences CD**” (Schank, 1975 [SHA75]) as well as free context grammars through “**Natural Language LN**” statements (Allen 1987 [ALL87] and Grosz 1986 [GRO86]) which are converted to the CD of consulting and reporting of methodologies, and CD for the automatic generation and programming for the Software Engineering Projects.

The methodology used for the development of the model of the SE-APIS is named KAMET (Knowledge Acquisition from Multiple Sources, Cairo, 1998 [CAI98]), with which was possible to guide each one of the phases of development, starting since the detection of the problematic, the design of the architecture of the model, to the construction of an operative PROTOTYPE of the SE-APIS, and the definition of a final model to built an integral expert system as a tool for the Software Engineering (HIS).

The programming of the SE-APIS was performed with the software CLIPS expert system development [CLI97] (C Language Integrated Production System) of the NASA Lyndon B. Jhonson Space Center, taking advantage of its programming objects oriented language facility (COOL) to design frameworks of knowledge, its potenciality defines facts and knowledge rules to process statements in natural language, and its flexibility to build CD that easily can be processed with the inference rules.

The SE-APIS has been evaluated in qualitative and quantitative form with several consultations and reports through sentences in natural limited language as well as with the automatic generation and calculation of the critical route of several software engineering projects. Even using the SE-APIS the self-evaluation was done for its phases of analysis, design and construction, obtaining succesfull results.

For future thesis, an overview is included as an addendum to this job, to develop an Expert System as an Integral Tool for Software Engineering (HIS), that should consider facilities of interconnection with other specific tools of CASE solutions, metric software for excellence in development, and its operation on internet environment.

# Capítulo I      Introducción

Actualmente, si bien es cierto que las Tecnologías de Información están brindando resultados satisfactorios en una gran cantidad de actividades de las diferentes especialidades del conocimiento humano, también es cierto que el incremento de estas tecnologías en muchas ocasiones ha despertado falsas expectativas para los usuarios que esperan una solución automática y de mayor calidad para sus problemas de procesamiento de información. Por otra parte no basta que las empresas cuenten con Tecnologías de Información sino que adicionalmente es necesario que cuenten con conocimiento especializado, de otra manera estas herramientas pasan a formar parte del gasto de producción de las empresas.

Este fenómeno también afecta a los especialistas de Desarrollo de Sistemas de Información, dado que muchos Ingenieros de esta especialidad no aplican de forma constante los métodos, herramientas y procedimientos de la Ingeniería de *Software* para lograr resultados de calidad satisfactoria para los usuarios. Aunado a este problema y como consecuencia de lo anterior no se realiza una buena planeación, programación y control de los proyectos de Ingeniería de *Software* y por lo tanto no se elaboran estadísticas ni métricas de desarrollo para poder emitir mejores estimaciones en el cálculo del tiempo, recursos necesarios y costo.

Es en este escenario que tiene su oportunidad un Sistema Experto que apoye a los Ingenieros de *Software*, dado que es posible que pueda aportar conocimiento especializado sobre conceptos de metodologías, administración de proyectos, estadísticas y métricas de desarrollo de *Software* así como modelos y métodos de los Sistemas que se estén desarrollando. Adicionalmente para incrementar la potencialidad de este tipo de Sistemas Expertos es factible realizar interfaces con herramientas de *Software* asistidas por computadora tal como los CASE e ICASE, que apoyan de forma automática y no automática la generación de diagramas de modelos, estructuras de bases de datos y lenguaje de código de programación.

Considerando este panorama se ha desarrollado en este trabajo de tesis un prototipo que muestra de forma básica como construir un Sistema Experto especializado en la Administración de Proyectos con apoyo de metodologías de Ingeniería de *Software*.

## **I.1. Problemática detectada en el desarrollo de Proyectos de Ingeniería de Software**

Es común dentro de las empresas que el desarrollo de sistemas de información no sea guiado adecuadamente con metodologías y estándares de desarrollo de *software*, produciendo sistemas que no cumplen con los requisitos de calidad para el usuario, por lo que tienen un alto riesgo de requerimientos de mantenimiento constante, acumulando un costo considerable a lo largo de su ciclo de vida.

Asimismo se ha detectado que la administración de proyectos de Ingeniería de *Software* es pobre en muchos casos (Olivia Niño, 2000 [NIÑ00]). Aún cuando las empresas han hecho grandes inversiones en *software* para administración de proyectos, los líderes de proyecto no realizan una adecuada estructuración, planeación y control de los mismos.

Al profundizar en los detalles de estos problemas fundamentales se puede observar que en las empresas no especializadas en el desarrollo de *software*, no se utilizan adecuadamente métricas de Ingeniería de *Software* en el desarrollo de sus proyectos.

En el caso particular del IMP existen algunas áreas especializadas de informática y líderes de proyecto que no utilizan métodos estandarizados, ni herramientas sistematizadas que les guíen mínimamente en forma de consulta y generación de informes acerca de metodologías de *software*, planeación y control de manera integral en el desarrollo y administración durante el ciclo de vida de sus proyectos.

### **Problemas en la Administración de Proyectos de Ingeniería de Software**

Se han realizado varios esfuerzos por identificar algunos de los principales problemas en los Proyectos de Ingeniería de *Software*, entre otros Thayer, Pyster y Wood [TPW02] y Olivia Niño [NIN00], han detectado algunos, que se observan en la tabla de la Figura I.1, y que pueden clasificarse dentro del dominio de la problemática de Administración de Proyectos, Metodologías y Métricas de desarrollo.

De acuerdo con Ian Sommerville [IAS02], las fallas de los Proyectos de Ingeniería de *Software* no se deben en la gran mayoría de los casos a la falta de experiencia de los programadores, sino al enfoque de administración de proyectos utilizado. Porque no es posible aplicar las mismas técnicas de administración derivadas de otras disciplinas a los Proyectos de Ingeniería de *Software*.

Los Proyectos de Ingeniería de *Software* deben ser soportados por metodologías, herramientas, procedimientos de administración y métricas estándar adecuados, que respondan a las siguientes preguntas, entre otras:

- a) ¿ Cuanto esfuerzo se requiere para completar una actividad?
- b) ¿ Cuánto tiempo calendario se necesita para completar una actividad?
- c) ¿Cuál es el costo total de una actividad?



NO.	PROBLEMAS DETECTADOS	DOMINIO DEL PROBLEMA
1	La planeación de proyectos es generalmente pobre	Administración de Proyectos
2	La capacidad para estimar correctamente los recursos requeridos para completar el proyecto es pobre	Métricas de desarrollo
3	No siempre existen procedimientos, métodos y técnicas para el diseño de un sistema de control del proyecto que permita a los ejecutivos controlar con éxito el avance de sus proyectos	Administración de proyectos Metodologías y Métricas de desarrollo
4	No existen estándares y técnicas para medir la calidad de desempeño y la cantidad de producción esperada de los programadores y analistas de procesamiento de datos	Métricas de desarrollo
5	Entrenamiento inadecuado sobre metodologías al personal de desarrollo	Metodologías de desarrollo
6	Falta de una metodología de administración de proyectos	Administración de proyectos
7	Los <i>milestones</i> del proyecto no están claramente definidos	Administración de proyectos Metodologías de desarrollo
8	Estimación inadecuada de los costos y el presupuesto del proyecto	Administración de proyectos Metodologías de desarrollo
9	Inexperiencia del Administrador del proyecto	Administración del proyecto
10	Requerimientos del sistema identificados inadecuadamente	Metodologías de desarrollo

Figura I.1. Problemas detectados en Proyectos de Ingeniería de *Software*

La Administración de Proyectos apoyada por Metodologías de desarrollo de sistemas es importante en proyectos de esta naturaleza, sin embargo al principio y durante la planeación detallada del proyecto se requiere de estimaciones de costos, presupuestos, y precios, de aquí que sea necesario utilizar Métricas de costos para el desarrollo de *software*, que considere los siguientes aspectos:

- a) Los costos de hardware y software incluyendo el mantenimiento de equipo
- b) Los costos de viajes y capacitación
- c) Los costos de pagos a ingenieros a partir de los esfuerzos realizados
- d) Los costos de la infraestructura de la empresa

Existen varios tipos de Métricas que permiten evaluar estos parámetros, entre ellas se encuentran las siguientes:

- Productividad de Putnam
- Modelos de algoritmos de costos, COCOMO II
- Tamaño y complejidad de la función, Puntos de Función

En el Capítulo II se tratarán diferentes herramientas que existen actualmente, y que apoyan el desarrollo de Proyectos de Ingeniería de *Software*, asimismo en el Capítulo III se describirán los conceptos básicos sobre la Administración de Proyectos, Metodologías y la Métrica del modelo de productividad de PUTNAM, algunos de los cuales forman parte del dominio del PROTOTIPO del Sistema Experto realizado en este trabajo de tesis.

## **I.2. Metodología de desarrollo de la solución**

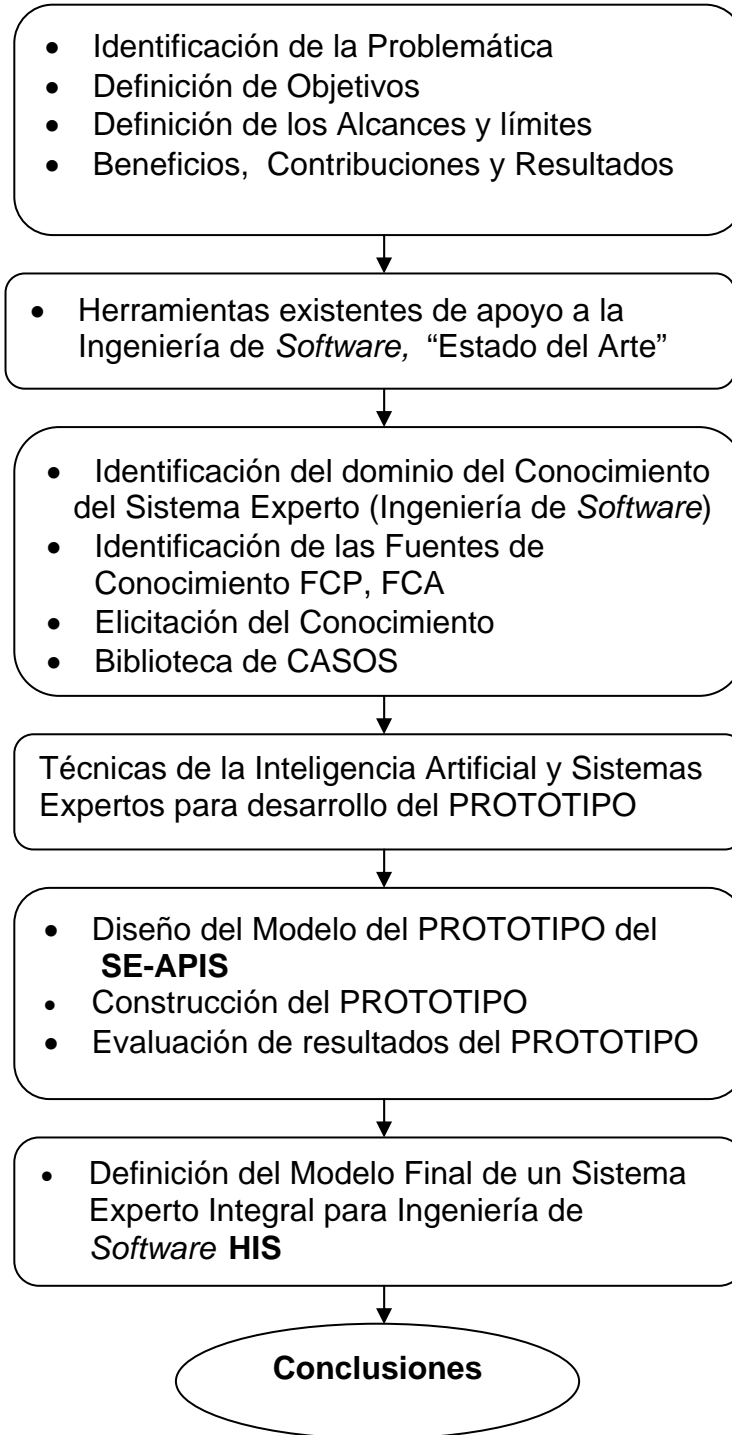
La metodología de desarrollo aplicada para construir un Sistema Experto que apoye a los líderes de proyecto en la Administración de Proyectos de Ingeniería de *Software* se ha estructurado partiendo de la adaptación de conceptos de la metodología KAMET, y ordenándolos con la aportación de los valiosos comentarios de los asesores de este trabajo tesis.

La metodología KAMET (Knowledge Acquisition from Multiple Sources, Cairo, 1998 [CAI98]), se sugiere como un conjunto de actividades para la adquisición de conocimiento para realizar una modelación de Sistemas Expertos más que una actividad de extracción o minería.

La metodología KAMET esta compuesta por cuatro etapas: planeación estratégica del proyecto, construcción del modelo inicial, construcción del modelo retroalimentado y construcción del modelo final.

Considerando que esta metodología no es una secuencia inflexible que debe seguirse a lo largo del desarrollo de los proyectos, se han tomado varios conceptos adaptándolos al desarrollo de un Sistema Basado en Conocimientos como Herramienta de apoyo para la Administración de Proyectos de Ingeniería de *Software*, y que se muestran en el diagrama de flujo de la Figura I.2.

**Figura I.2. Metodología de Desarrollo del Sistema Basado en Conocimiento como Herramienta de apoyo Para la Administración de Proyectos de Ingeniería de Software HIS, SE-APIS**



### I.3. Objetivos y Alcances

#### Objetivos

El objetivo principal que se persigue en este trabajo de tesis es analizar, diseñar y construir un PROTOTIPO de un Sistema Experto Basado en Conocimiento (*Knowledge Based Expert System*) para apoyo a la Administración de Proyectos de Ingeniería de *Software* (**SE-APIS**), que sirva como herramienta inteligente a los Ingenieros de esta especialidad, fundamentalmente en las fases de estructuración, planeación y programación de proyectos y asimismo, les brinde consultoría en línea sobre metodologías de Ingeniería de *Software*.

Este prototipo de Sistema Experto debe permitir al Ingeniero de *Software* contar con información suficiente en línea sobre los conceptos de las diferentes etapas de las metodologías de sistemas. Lo anterior podrá realizarlo con facilidad por medio de una interfaz Hombre – Máquina en Lenguaje Natural, que le permitirá consultar y obtener reportes de cada una de las fases y actividades de la metodología requerida.

El Ingeniero de *Software* apoyado mediante esta Sistema Experto contará con una planeación adecuada desde el inicio del ciclo de vida de sus proyectos. A partir de la creación de proyectos en forma automática que se construirán con una estructura WBS (*Work Breakdown Structure*) conteniendo fases de desarrollo, y a su vez cada fase generada incluirá sus actividades correspondientes.

Después de la creación del proyecto, el Ingeniero de *Software* por medio de una oración en Lenguaje Natural podrá programar la red de fases formada con precedencias previamente almacenadas en la Base de Conocimientos. Esta programación del proyecto incluirá el cálculo de la ruta crítica de la red mediante un algoritmo inteligente.

El Sistema Experto como apoyo a la administración de proyectos de Ingeniería de *Software*, no desarrolla de manera extensa la generación de reglas de conocimiento sobre métricas de *software*, sin embargo abre esta posibilidad al incluir el modelo de PUTNAM [PUT78] para el cálculo de la productividad de proyectos complejos.

Adicionalmente con la elaboración de este trabajo se busca establecer los modelos básicos para integrar herramientas CASE's existentes que sirven como apoyo a la generación de los diferentes modelos y métodos utilizados en el desarrollo de *software*, con herramientas de un contexto más amplio como son las metodologías y la Administración de proyectos que llevaran a los líderes de Proyectos de Ingeniería de *Software* a obtener mejores resultados.

Finalmente existe la posibilidad de que este modelo pueda servir como base para trabajos futuros de tesis en la construcción de un Sistema Experto Integral de Ingeniería de *Software*.

## Alcances

En el trabajo preliminar de la elaboración de este tema de tesis se había planteado desarrollar una Herramienta de apoyo a la Ingeniería de *Software* a la cual se le denominó **HIS**. Sin embargo conforme se ha ido profundizando en las metodologías, los procesos y las herramientas que se deben integrar en un ambiente de esta naturaleza, para brindar apoyo de forma inteligente a los Ingenieros de esta especialidad, se observó que llevaría más tiempo y recursos humanos especializados que los necesarios para realizar un trabajo de tesis.

Es debido a la razón anterior que en este trabajo de tesis se han enfocado los esfuerzos al desarrollo de un PROTOTIPO de Sistema Experto que brinde apoyo a la Administración de proyectos de Ingeniería de *Software*, al cual se le ha denominado **APIS**.

En la Figura 1.3 se tiene representado el alcance de un Sistema Experto Integral para la Ingeniería de *Software*, así como el alcance del PROTOTIPO del SE-APIS el cual está delimitado por un recuadro de línea interrumpida. A continuación se presenta la descripción general de los dos alcances:

### **Alcance de un Sistema Experto Integral para la Ingeniería de *Software***

El modelo de una Herramienta Integral para la Ingeniería de *Software* de manera completa debe estar estructurado por varios módulos desarrolladas por medio de técnicas de Inteligencia Artificial, siendo estos los siguientes

- Adquisición de Conocimientos
- Base de Conocimientos de Metodologías
- Base de Conocimientos de Métodos, Modelos y Métricas
- Base de Conocimientos de Proyectos de Ingeniería de *Software*
- Base de Conocimientos de desarrollos de *Software* y Sistemas de Información
- Analizador de parámetros y coeficientes estadísticos
- Interfaz con el usuario por medio de Lenguaje Natural, Menús funcionales e Internet
- Reglas de inferencia para solución de procesos y control del Sistema Experto

Debido a la complejidad que cada una de estos módulos representa, tal como se explica de manera más amplia en el apartado de trabajos futuros, para poder formar una Herramienta Integral para la Ingeniería de *Software*, en este trabajo de tesis solo se han considerado los módulos básicos para poder desarrollar un prototipo de un Sistema Experto para la Administración de Proyectos de Ingeniería de *Software*.

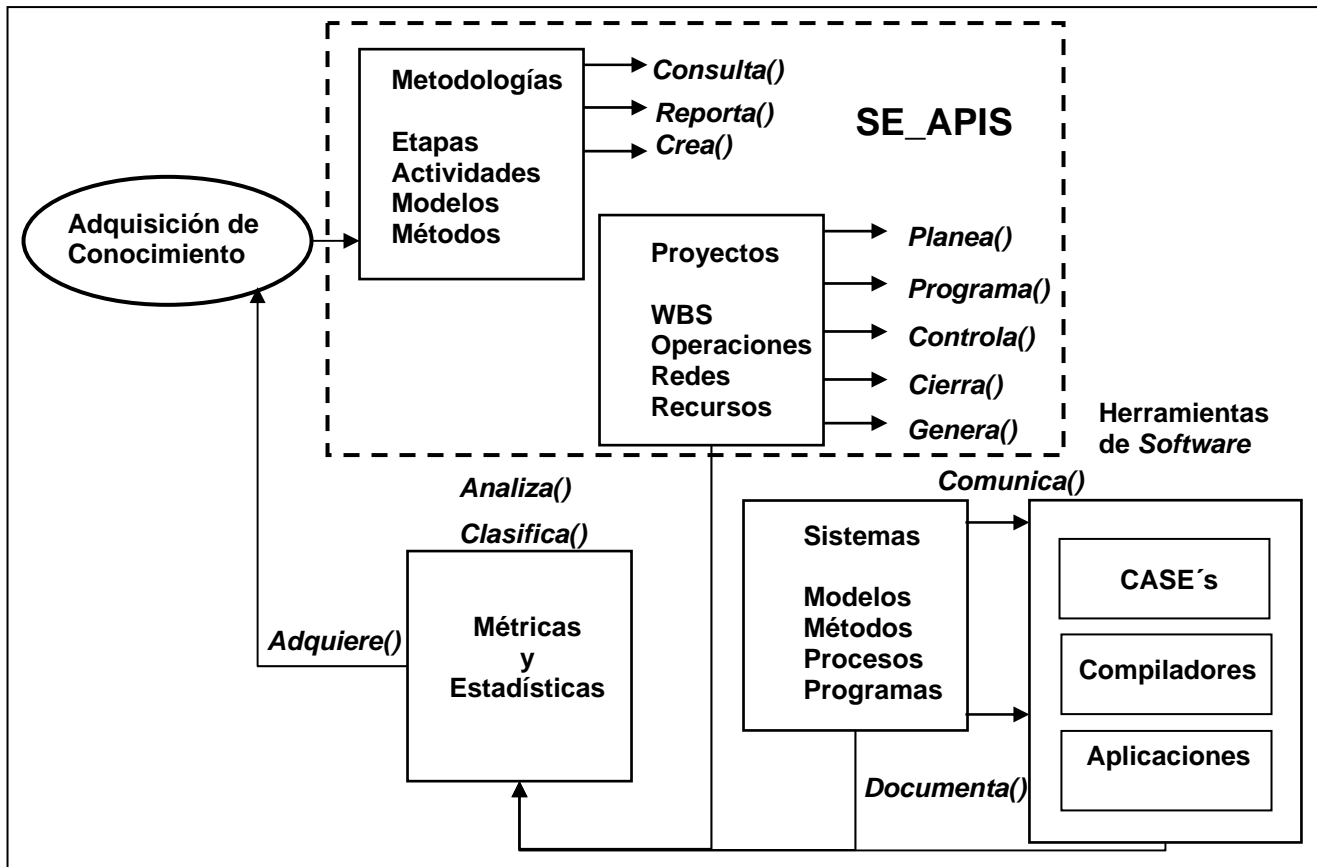


Figura I.3.- Modelo de un Sistema Inteligente Integral HIS y del PROTOTIPO SE-APIS

### Alcance del PROTOTIPO del Sistema Experto para la Administración de Proyectos de Ingeniería de Software (SE-APIS)

El alcance desarrollado en este trabajo de tesis es un PROTOTIPO que incluye las funciones básicas para que un Ingeniero de *Software*, pueda consultar y reportar conocimientos sobre metodologías de desarrollo, sus etapas y actividades. Asimismo pueda generar proyectos de Ingeniería de *Software* a partir de una base de conocimientos de metodologías, realice la programación del proyecto y obtenga el cálculo de la ruta crítica de la red de fases del ciclo de vida del Sistema de Información a desarrollar. Aún que de forma incipiente se podrán utilizar métricas para realizar cálculos de horas requeridas en un proyecto por medio del método de PUTNAM. Las funciones de este PROTOTIPO podrán ser utilizadas por el usuario a través de una interfaz de Lenguaje Natural. Esta funcionalidad del Sistema Experto esta desarrollada en los siguientes módulos básicos:

- Base de Conocimientos de Metodologías
- Base de Conocimientos de Proyectos de *Software*
- Interfaz con el usuario por medio de Lenguaje Natural
- Reglas de inferencia de Proceso y Control

El desarrollo de estos módulos se llevo a cabo mediante técnicas de Inteligencia Artificial programadas en el motor de inferencia CLIPS (C Lenguaje Integrated Production System [CLI97]) de la NASA.

## I.4. Beneficios y Contribuciones

### Beneficios

Dentro de los beneficios que traerá el desarrollo del Sistema Experto para la Administración de proyectos de Ingeniería de *Software*, se han percibido los siguientes:

- Contar con una herramienta que pueda aplicarse en la actualización de los Ingenieros de *Software* en el uso de metodologías de desarrollo.
- Iniciar con la difusión para el manejo de estándares de metodologías de desarrollo de *software* en el IMP.
- Desarrollar pruebas con datos reales para apoyar la estructuración, planeación y programación de proyectos de Ingeniería de *Software*.
- Contar con una alternativa para reducir el tiempo y costo en la elaboración de propuestas de desarrollo de productos de *software*.
- Promover el registro de estadísticas e información sobre el comportamiento de los proyectos y el uso de métricas para el desarrollo de *software*.

### Contribuciones

Las contribuciones visualizadas a partir del desarrollo de este trabajo de tesis son las siguientes:

- Aplicación de técnicas de la Inteligencia Artificial en el desarrollo de un modelo prototipo INICIAL para la Administración de Proyectos de Ingeniería de *Software*.
- Planteamiento de un modelo inteligente para resolver la problemática de los Proyectos de Ingeniería de *Software*.
- Definición de las características y arquitectura general de un modelo FINAL integral para la Ingeniería de *Software* (HIS).
- Recopilación bibliográfica de diversas metodologías de la Ingeniería de *Software* y de la Inteligencia Artificial.

## I.5. Motivación y Justificación de desarrollo

### Motivación

La motivación inicial para realizar este trabajo surgió al observar que las herramientas CASE facilitan el desarrollo de modelos de sistemas de información y se apoyan en bases teóricas metodológicas, desde métodos clásicos como el *análisis y diseño estructurado*, hasta conceptos modernos como *programación automática y programación orientada a objetos (POO)*, que el desarrollador utiliza de acuerdo con sus conocimientos sobre estas metodologías.

Considerando lo anterior, surge la idea de desarrollar una herramienta que permitiera al Ingeniero de *Software* contar con un ambiente integral de desarrollo que manejará varias metodologías, sus fases y actividades interconectadas con herramientas CASE's.

La siguiente motivación surge al realizar estudios de Inteligencia Artificial que me permitieron establecer un modelo que integrará conocimientos sobre Proyectos y Metodologías de *Software*, y que pudiera ser explotado por medio de Lenguaje Natural.

La tercera motivación es haber detectado la problemática de los Líderes de Proyecto para estructurar adecuadamente sus trabajos de desarrollo de *software*, la falta de uso común de metodologías y métricas, y la inexistencia de bancos de datos estadísticos de los sistemas y *software* desarrollado.

El desarrollo de este trabajo de tesis me ha permitido seguir avanzando en el diseño de un Sistema Experto Integral para la Ingeniería de *Software* (HIS), sin embargo por lo amplio del tema y por la complejidad técnica que involucra el desarrollo de un sistema con este alcance, este trabajo de tesis únicamente se ha limitado al desarrollo de un PROTOTIPO para apoyo a la Administración de Proyectos de Ingeniería de *Software* (SE-APIS)

### Justificación de desarrollo del SE-APIS

Los problemas del desarrollo de *software* y de sistemas de información, no son exclusivos de los Ingenieros de *Software* sino que también participan desarrolladores que no son ingenieros especialistas en la materia, realizando sistemas con muy poco conocimiento de Metodologías de Desarrollo de Sistemas, Administración de Proyectos y Métricas de Ingeniería de *Software*.

En el IMP, dentro de la competencia de Tecnologías de Información actualmente existen aproximadamente 306 especialistas activos y 100 asistentes y becarios, que se dedican al desarrollo de *software* y sistemas de información, para consumo interno y para nuestro principal cliente PEMEX. La participación de este personal dentro de los proyectos, es desde Líderes de proyecto, Ingenieros de *Software*, especialistas de Tecnologías de Información, programadores de código y asistentes de informática.



Este trabajo de tesis pretende dar la pauta para iniciar la difusión y apoyo sobre el conocimiento de estándares de Metodologías, Administración de Proyectos y Métricas, entre el personal de la competencia de Tecnologías de Información y especialidades afines del IMP, con el objetivo de coadyuvar a la construcción de sistemas de calidad, incrementar la productividad en su desarrollo y disminuir los costos de mantenimiento.

Para realizar las expectativas anteriores se desarrollará dentro de este trabajo de tesis un PROTOTIPO de un Sistema Experto como apoyo a la Administración de Proyectos SE-APIS, que será el punto de partida para la construcción de una Herramienta Integral de Ingeniería de *Software*, lo cual puede ser motivo de un trabajo futuro de tesis.

El desarrollo de este PROTOTIPO ayudará a los Líderes de Proyecto de Ingeniería de *Software* en forma original con la aplicación de métodos de **Inteligencia Artificial (IA)** que permiten construir una base de conocimientos a partir de *Marcos*, y la definición de Dependencias Conceptuales(CD), para consultar y reportar fases y actividades de Metodologías así como generar Proyectos de Ingeniería de *Software* en forma automática. Asimismo se diseñaran y construirán las reglas de inferencia para el cálculo de la ruta crítica mediante el algoritmo CPM. La comunicación con los usuarios se construirá con una gramática libre de contexto limitada con estructura específica, que permite interpretar frases en *Lenguaje Natural* que el usuario utilizará como interfaz y que a partir de ellas generen un lenguaje destino CD para consulta() reporta(), crea() y programa(), y que puedan ser parte de las reglas de inferencia del SE-APIS.

La implantación del SE-APIS junto con el motor de inferencia *CLIPS* en el que fue desarrollado requiere de recursos de cómputo mínimos para su operación y puede ser instalado en cualquier equipo PC de cómputo que opere con el sistema operativo *windows*.

Dado que el Software *CLIPS* en el que está desarrollado el SE-APIS está incorporando tecnologías de información por medio de JAVA WEB y Oracle su ambiente de operación puede ser mejorado notablemente.

Se espera que el SE-APIS desde el principio de su implantación logre contribuir al desarrollo de Sistemas de Información en el Instituto Mexicano del Petróleo, al incorporar una forma inteligente de consultar y reportar conocimientos sobre Metodologías de Sistemas de Información y la generación automática de Proyectos para la Administración de Ingeniería de *Software*. Y que por otra parte, sea una contribución para el desarrollo futuro de una Herramienta Integral Inteligente para la Ingeniería de *Software* al construirse Interfaces con otros Sistemas Expertos.

## I.6. Resultados esperados

Con el fin de verificar y probar los resultados que debe proporcionar el SE-APIS, se propone realizar varias consultas y reportes de información referente a Procedimientos, Administración, Metodologías de Ingeniería de *Software*, sus etapas y actividades, así como la generación automática de la estructura de un proyecto estándar para el desarrollo de un sistema de información.

En la ejecución de la CD consulta(), se desea que el SE-APIS interprete oraciones en lenguaje natural con la estructura “**actor consulta complemento**”, y a partir de esta construya la CD y active reglas de inferencia que respondan con el despliegue de la explicación adecuada para diversas instancias de conocimiento de procedimientos, metodologías, administración, proyectos y etapas.

Para la CD reporta(), el SE-APIS debe reconocer las frases “**actor reporta complemento**”, construya la CD correspondiente y despliegue en la pantalla del usuario cada una de las etapas y sus características sobre la Metodología para desarrollo de sistemas de información CADM.

En la ejecución de la CD crea(), el SE-APIS debe interpretar frases “**actor crea complemento1 complemento2**”, construir la CD, activar las reglas de inferencia que preguntan interactivamente al usuario los datos generales del proyecto, a partir de los cuales el sistema tomará datos de las instancias de las etapas y actividades de la metodología CADM, para generar de forma automática las fases y operaciones del proyectos del sistema de información a desarrollar. Adicionalmente el sistema debe generar los costos, duración y recursos en horas necesarias para desarrollar cada una de las fases del sistema de información. Así mismo a partir de la métrica del modelo de PUTNAM el SE-APIS debe emitir el estimado de líneas de código que se programarán y que son el equivalente a la cantidad de horas de esfuerzo (trabajo total del proyecto), aplicado a tres posibles ambientes de desarrollo utilizado: sin metodología, con metodología y con herramientas CASE. Los hechos y datos generados automáticamente en los proyectos deben quedar almacenados en archivos planos con estructuras de Marcos definidas dentro de la Base de Conocimientos. Toda la información obtenida en la generación del proyecto debe ser desplegada en el monitor del usuario.

Utilizando la CD programa, el sistema debe interpretar frases en lenguaje natural de la forma “**actor programa proyecto**” y generar la CD para activar las reglas de inferencia que realicen la programación del proyecto con el algoritmo de ruta crítica. El resultado del cálculo de la programación del proyecto debe desplegarse en el monitor del usuario.

Invariablemente cualquier consulta o reporte que se desee hacer al SE-APIS debe tener la estructura gramatical “**actor verbo complemento**”, la generación de proyectos en forma automática debe tener la estructura “**actor verbo complemeto1 complemento2**” y la programación de proyectos la estructura “**actor verbo proyecto**”.

## I.7. Descripción del contenido del trabajo de tesis

El Sistema Basado en Conocimiento como Herramienta de apoyo para la Administración de Proyectos de Ingeniería de *Software* HIS (SE-APIS), es un Sistema Inteligente que trata de proporcionar una herramienta de *software* que apoye al Ingeniero de esta especialidad en la solución de la problemática de administrar proyectos, motivo por lo cual se ha realizado en este tema de tesis el modelo de un PROTOTIPO que proporcione la información mínima sobre metodologías de desarrollo y ayude a crear y programar proyectos de *software* con una estructura bien definida. Por otra parte el desarrollo del SE-APIS permitirá dejar un planteamiento para desarrollar una Herramienta Integral para la Ingeniería de *Software*.

En el capítulo I de este trabajo de tesis, se presenta la descripción de la problemática sobre administración de proyectos en la que viven inmersos los Ingenieros de *Software*, para lo cual se propone como solución el desarrollo del SE-APIS. Este desarrollo está definido en base a la metodología KAMET (Knowledge Acquisition from Multiple Sources), e incluye como parte de su secuencia de actividades: la definición de objetivos y alcances, la descripción de beneficios y contribuciones, la identificación del dominio y las fuentes de conocimiento del SE-APIS, así como el diseño, la construcción y las pruebas que deben de realizarse en el desarrollo de un Sistema Experto.

Dentro del capítulo II, se presenta el “Estado del Arte” de las Herramientas existentes de apoyo a la Ingeniería de *Software*, entre las cuales se incluye: *software* para la Administración de Proyectos, diferentes Herramientas CASE que apoyan al desarrollo más expedito de sistemas de información, así como varias perspectivas de automatización con potentes herramientas inteligentes, como SE-MeRCI para reingeniería de *software* y el ICASE para diseño automático de sistemas de información con reuso de diseños. En el último apartado de este capítulo se presenta una tabla comparativa de los diferentes Sistema Expertos descritos.

En el capítulo III, se describe el dominio de conocimientos del SE-APIS el cual incorpora el Marco Teórico de la Ingeniería de *Software*, incluyendo las diferentes Metodología y ciclos de vida del desarrollo de *Software* y Sistemas de Información. Aunados a estos conceptos se presenta un marco teórico con las mejores técnicas actuales para la Administración de Proyectos emitidas por el Project Management Institute **PMI**. Asimismo debido a la importancia que tienen las métricas de *Software* en el desarrollo eficiente de proyectos de este tipo, se incluye dentro de este dominio de conocimientos la ecuación de PUTNAM para el cálculo de la productividad de *software*. Todos estos conceptos pueden integrarse de forma natural en un Modelo de Base de Conocimientos de Ingeniería de *Software*.

En el capítulo IV, se realiza la identificación y la elicitación de las Fuentes de Conocimiento Pasivas (FCP) y de las Fuentes de Conocimiento Activas (FCA) a partir de las cuales se plantea la construcción de una Biblioteca de Casos, incluyendo el caso de la metodología CADM (CASE Application Development Method [DES00]), para ser incorporada en la base de conocimientos del dominio del SE-APIS.

El Capítulo V, se describen el grupo de técnicas de Inteligencia Artificial que sirven de apoyo a la modelación y construcción del SE-APIS, entre estas técnicas se encuentran: la representación del conocimiento, las estructuras de ranura y relleno débil “**Marcos de**

**Minsky**", así como las estructuras de ranura y relleno fuerte "Dependencias Conceptuales" (**CD**) definidas por **Schank** en 1975. Estas estructuras asociadas al procesamiento de Lenguaje Natural permiten brindarle al usuario una interfaz para comunicarse de una forma sencilla con el sistema.

Utilizando las técnicas de Inteligencia Artificial, en el capítulo VI, se diseña el Modelo de estructuras de *Marcos* de Conocimiento y el Modelo de procesamiento del SE-APIS. Dentro del Modelo de *Marcos* se consideran dos grupos de Clases: las de metodologías de desarrollo y las de Administración de Proyectos. En el Modelo de procesamiento se tiene la consulta y reportes sobre metodologías, la creación de proyectos y la métrica de Putnam, así como la programación de la red del proyecto creado. En este capítulo también se define el Modelo de la interfaz con el usuario por medio del procesamiento del Lenguaje Natural, considerándose de mucha importancia la estructuración de las CD consulta(), reporta(), crea() y programa(), dentro del analizador semántico.

En el capítulo VII, tomando como base los Modelos del SE-APIS se describe la construcción y programación de las estructuras de conocimiento y las reglas de inferencia del SE-APIS, utilizando la herramienta de desarrollo de Sistemas Expertos *CLIPS* (C Lenguaje Integrated Production System).

El diseño de pruebas, la forma de realizar la ejecución del SE-APIS dentro de la herramienta *CLIPS* y la evaluación de los resultados obtenidos para varias pruebas de consultas y reportes sobre metodologías de *software*, así como la generación y programación de un proyecto que permite realizar una autoevaluación del SE-APIS, se presentan en el capítulo VIII. En este punto, es conveniente mencionar que los resultados obtenidos durante las diferentes pruebas, incluyendo las que no están en este documento de tesis han resultado satisfactorios.

En el capítulo IX, se presentan los trabajos futuros que pueden ser desarrollados a partir del PROTOTIPO del SE-APIS, fundamentalmente se plantea el desarrollo de una Herramienta Integrada de apoyo a Ingeniería de *Software* (HIS), que a partir de la estructura del SE-APIS incorpore métricas y estadísticas que apoyen el análisis de proyectos, se construya un módulo de adquisición de conocimiento y un módulo de interfaz con herramientas CASE. Este sistema HIS debe de operar en un ambiente Internet para que funcione como una WEB inteligente.

Finalmente, se presentan las conclusiones de este trabajo de tesis, así como la relación de los anexos que se incluirán en un disco compacto que se adjuntará al documento de la tesis. Dentro de estos anexos se almacenarán los programas completos del SE-APIS desarrollados en *CLIPS*, la metodología CADM de la compañía Oracle y las presentaciones realizadas sobre la tesis.

## Capítulo II Herramientas existentes de apoyo a la Ingeniería de Software “Estado del Arte”

Existen actualmente en el mercado o en etapa de investigación, *software* que apoya en diferentes formas al Ingeniero de *Software*, incluyendo herramientas genéricas para la Administración de Proyectos, Métricas de desarrollo, CASE e I-CASE, cada una de las cuales apoya el desarrollo y la administración en la construcción de nuevo *software*, paquetes de aplicación o sistemas de información.

### II.1. *Software* para la Administración de Proyectos

En la actualidad existen diferentes paquetes de *software* para la Administración de Proyectos, entre otros productos comerciales se encuentran los siguientes:

- *Project System PS* de la compañía SAP R/3 [PSP99]
- *Primavera Enterprise* [PEN02]
- *Microsoft Project* [MSP02]
- *Artemis Project View* [APV02]
- *Project BAAN*

Cada uno de ellos brinda interfaces que tratan de facilitar la interacción con la computadora a los Líderes de Proyecto. Asimismo, todos ellos se basan en las etapas fundamentales para la administración de proyectos: Inicio, Planeación, Ejecución, Control y Cierre.

En general dentro de las funciones que cubren los paquetes de *software* para la administración de proyectos, se identifican las siguientes:

- Interfaces gráficas drag and drove que permiten definir las funciones personalizadas que ocupará el Líder de proyecto, dentro de su proceso de Administración de proyectos, dependiendo de los requerimientos de la empresa.
- Estructuración del Alcance y de la WBS del proyecto para llevar un mejor control de desarrollo.
- Interfaces gráficas para diseñar la red de actividades y cálculo de la ruta crítica del proyecto.
- Planeación completa de los diversos aspectos del proyecto en cuanto a tiempo, recursos y costos.
- Asignación de los recursos humanos que participan en el proyecto.
- Determinación del presupuesto del proyecto.
- Seguimiento de avance real del proyecto considerando sus diferentes parámetros: trabajo planeado y real, costos registrados y presupuesto ejercido en la ejecución del proyecto.
- Consulta e impresión de reportes para diferentes rubros del proyecto con diversos criterios de selección.
- Presentación de varios tipos de gráficos para representar el avance y estadísticas tanto real como planeado para recursos, costos y fechas del proyecto.

- Herramientas para la evaluación y la conclusión de las actividades tanto operativas como financieras del proyecto.
- Registro de la facturación tanto de egresos como de ingresos del proyecto.
- Manejo de control de estatus para ir controlando las diferentes fases del proyecto: inicio planeación, ejecución, control y cierre.
- Adicionalmente algunos de los paquetes de Administración de Proyectos incluyen un ambiente analítico para informar a la Alta Dirección el estado en que se encuentran los proyectos de la empresa, apoyando de esta forma la toma de decisiones.

En la Figura II.1 se presenta una comparación de las diferentes características de los paquetes de *software* para la Administración de Proyectos.

No.	Característica	PS-SAP	Prima vera	MS- Project	Artemis	BAAN
1	Diseño personalizado del proceso	X				X
2	Definición de la estructura del proyecto	X	X	X	X	X
3	Definición de las actividades del proyecto	X	X	X	X	X
4	Planeación de los recursos materiales y servicios del proyecto	X	X	X	X	X
5	Planeación de los recursos humanos del proyecto	X	X	X	X	X
6	Tabla gráfica de planeación de la estructura y red del proyecto	X	X	X	X	
7	Integración de datos de forma empresarial ERP	X				
8	Simplicidad en la Interfaz de registro de datos con el usuario		X	X	X	
9	Generación de reportes agregando campos adicionales a un reporte inicial con datos estándar	X	X		X	
10	Generación sencilla de reportes		X	X	X	
11	Registro de costos del proyecto	X	X		X	X
12	Manejo de control presupuestal	X	X		X	
13	Registro de los ingresos del proyecto y datos del cliente	X				
14	Análisis de avance del proyecto	X	X	X	X	
15	Facilidad gráfica de análisis		X	X	X	
16	Interfaz vía Internet	X	X	X	X	X
17	Control de estatus del proyecto	X				
18	Análisis de disponibilidad de recursos	X	X		X	
19	Facilidad de análisis de Ejecutivos		X		X	
20	Control de horas del personal	X	X		X	

Figura II.1. Comparación de características de *software* para Administración de Proyectos

Aún cuando los productos para la Administración de Proyectos, cuentan con varias facilidades para los Líderes de Proyecto. Este tipo de *software* no ha logrado completamente los objetivos para lo cual fue creado. Convirtiéndose más en un negocio de los fabricantes de *software*, que en una solución totalmente efectiva para las empresas. Dado que estas realizan grandes inversiones en recursos, no solo en Tecnologías de Información sino también en la capacitación de Técnicas para la Administración de Proyectos para el personal.

El problema fundamental se encuentra en que este tipo de herramientas requieren de bastantes habilidades para su manejo y muchos de los Líderes de proyecto no están dispuestos a sacrificar tiempo de desarrollo tecnológico por el tiempo de analizar y elaborar la parte administrativa de los proyectos. Por esta misma razón los paquetes de *software* en cuanto más potentes y más funciones incluyen en sus características provocan una mayor confusión en su manejo sobre todo cuando los Líderes de Proyecto son inexpertos o reacios al uso de Tecnologías Informáticas.

Por otra parte al ser un *software* de aplicación genérica para la Administración de Proyectos ,no incorporan información especializada de acuerdo al tipo de proyecto que se esta realizando, por lo que le dejan toda la definición de la estructura especializada al Líder del Proyecto, quién en muchos casos no la registra completa y oportunamente, teniendo consecuencias graves en la toma de decisiones de las empresas.

La inclusión de técnicas de Inteligencia Artificial que incorporen bases de conocimiento especializada como por ejemplo de Ingeniería de *Software*, podría ayudar a tener conocimiento en línea sobre la especialización técnica de los proyectos por desarrollar, adicionalmente un generador de proyectos a partir de metodologías de desarrollo, tendría la tarea de construir proyectos bien estructurados y fácilmente manejables con la combinación de una interfaz de Lenguaje Natural y menús funcionales.

## II.2. Herramientas CASE

Actualmente existe un amplio rango comercial de CASE (Ingeniería de *Software* Asistida por Computadora) con diferentes puntos de enfoque, que se utilizan para ayudar al Ingeniero de *Software* en sus actividades de desarrollo, tales como el análisis de requerimientos, el modelado de sistemas, la depuración y pruebas [IAS02]. En la actualidad, todos los métodos utilizan tecnología CASE asociada, como por ejemplo editores para las notaciones utilizadas en los métodos, módulos de análisis que verifican que el modelo del sistema este acorde con las reglas del método y generadores de informes que ayudan a crear la documentación de los sistemas. Las herramientas CASE también incluyen generadores de código fuente automático a partir del modelo del sistema.

Existen herramientas CASE de alto nivel cuyo propósito es ayudar a analizar y diseñar en las fases iniciales del proceso del *software*. Las herramientas CASE que están diseñadas para ayudar a la implantación y pruebas, como los depuradores, sistemas de análisis de programas, generadores de casos de prueba y editores de programas, se les conoce como herramientas de bajo nivel.

Los CASE están formados por bancos de trabajo [IAS02], que son un conjunto de herramientas que ayudan a una fase particular del proceso del *software* como el diseño, la

implantación o las pruebas. La ventaja de agrupar las herramientas CASE en un banco de trabajo, es que estos puede trabajar de forma conjunta para suministrar una ayuda más completa. Los servicios comunes se implementan y son llamados por todas las demás herramientas. Las herramientas del banco se integran mediante archivos o estructuras de datos compartidas.

Los bancos de trabajo de análisis y diseño están estructurados para apoyar el modelo del sistema durante la etapa de análisis y diseño del proceso del *software*. Estos comúnmente soportan un método como el análisis y diseño orientado a objetos, aunque alternativamente, existen sistemas de edición de diagramas más generales que utilizan varios métodos.

La Figura II.2 muestra las herramientas que se pueden incluir en un banco de trabajo para el análisis y diseño. Estas herramientas habitualmente se integran por medio de un depósito compartido que son propiedad del vendedor de dicho banco, y por lo tanto estas herramientas son normalmente entornos cerrados.

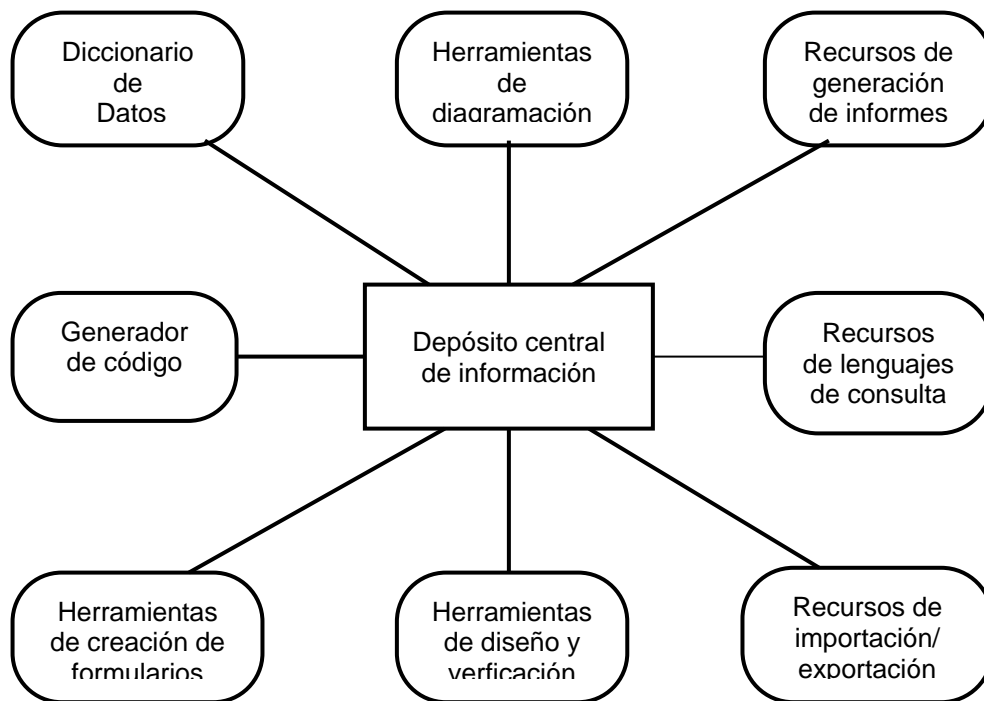


Figura II.2. Un banco de trabajo para el análisis y diseño

En algunos casos, es posible generar un programa o un fragmento de este a partir de la información suministrada en el modelo del sistema. Los generadores de código que se incluyen en los bancos de trabajo para el análisis y diseño generan código en lenguajes como Java, C++ o C. Puesto que los modelos excluyen detalles de los niveles inferiores, al generador de código no le es posible generar el sistema completo. Habitualmente se requiere codificar a mano para completar los programas.

Algunos bancos de trabajo CASE para el análisis y diseño ayudan a los métodos para desarrollar aplicaciones de sistemas de negocios. La plataforma de desarrollo y la de aplicación normalmente son la misma de tal forma que se utiliza un sistema de base de datos como Oracle para implementar el depósito compartido de la herramienta. Estos CASE



incluyen muchas de las características de un entorno de programación 4GL por lo que en lugar de generar un lenguaje de programación convencional para el diseño, generan un lenguaje de base de datos 4GL.

### **II.2.1. CASE para diseño gráfico por computadora +1 *Software Engineering***

Un ejemplo de un banco de trabajo CASE es el de la compañía +1 *Software Engineering* [1SE02], es un conjunto de herramientas para Diseño Asistido por Computadora (CAD) para estaciones de trabajo Sun, sistema operativo Solaris, y front end Open Windows y Motif con generación de código HTML para informes. Las tareas que ejecutan cada una de las herramientas en el proceso de CAD son las siguientes:

- Administrador de tareas en los proyectos de CAD (+1CM Fact Sheet), controla los accesos de múltiples usuarios trabajando simultáneamente en el mismo proyecto.
- Controlador de Métricas de trabajo (+1 Metrics4Project Fact Sheet), permite llevar un control estadístico de los trabajos que se están realizando, actualizados, en espera de autorización, autorizados, cerrados y no aprobados.
- Controlador de tareas (+1CR Fact Sheet), soporta una gran cantidad de reportes de problemas para brindar apoyo en la administración del proceso de diseño, cambios del número de requerimiento, versiones de información de estado del diseño, manejo de prioridades de diseño, detalles de descripción de problemas y acciones correctivas de diseño.
- Diccionario de datos (+1 Data Tree Fact Sheet), almacena la información sobre elementos de datos, estructuras de datos, archivos, entidades externas, glosario de términos y datos de usuario.
- Generador de reportes (+1 Reports Fact Sheet), proporciona todo tipo de informes acerca del proceso de diseño: tablas de referencia cruzada, encabezados de archivos, métricas de *software*, solución a problemas comunes, conclusiones, problemas, etc.
- Reuso de diseños (+1 Reuse Fact Sheet), soporta la generación de nuevos diseños a partir del reuso de un repositorio de datos, permite usar todo el código y la documentación disponible.
- Ingeniería reversa (+1 ReverseC Fact Sheet), lee los archivos de diseño y el código existente en C, lo analiza y genera el ambiente de archivos necesarios para el modelado.

### **II.2.2. Designer 2000 de Oracle**

Designer/2000 [DES00] sigue la estructura modular tradicional, contiene un repositorio implementado en la base de datos relacional de Oracle. El repositorio consiste de tablas que almacenan información sobre el sistema que se está analizando, diseñando y produciendo. Como el repositorio es una base de datos estándar de Oracle, tiene los beneficios y consideraciones de un sistema multiusuario: seguridad, conectividad y concurrencia.

Esta herramienta CASE soporta un amplio rango de tareas, debido a que es una herramienta integrada que considera tanto el nivel alto como el nivel bajo del proceso de desarrollo, porque comprende desde las fases de estrategia, análisis y diseño, así como la generación de código SQL de 4GL en la fase de construcción.

Proporciona al Ingeniero de *Software* cinco diagramadores y utilidades de desarrollo del sistema: Modelador de procesos de negocio, Modelador de sistemas, Diseñador de sistemas, Generadores y Utilidades de repositorio incluyendo expertos de diseño.

Dentro del Modelador de sistemas tiene una interfaz gráfica para realizar los diagramas de jerarquías de funciones, diagramas entidad relación, diagramas de flujo de datos. Los Generadores además del código SQL, generan los esquemas de formas de pantallas para el usuario, estructuras de reportes y gráficas así como código HTML para ambiente Internet.

### II.2.3. CASE *Rational Rose*

*Rational Rose* [CRR02] es un modelador integrado que sirve como herramienta para el desarrollo de *software* usando el estándar UML (Unified Modeling Lenguaje) para manejo de objetos. Habilita a todos los miembros del equipo de desarrollo para comunicarse de manera colaborativa para desarrollar un mejor producto.

La utilización de las herramientas CASE no es muy común en las empresas debido a que sus costos de licenciamiento son altos. Asimismo aún cuando las empresas cuenten con este tipo de herramientas no han podido resolver la problemática de los Líderes de proyecto debido a que no integran herramientas para la Administración de Proyectos estructurados adecuadamente de acuerdo con el conocimiento de las metodologías de Ingeniería de *Software*.

## II.3. Sistemas Inteligentes para la Ingeniería de *Software*

Dentro del ámbito de la Inteligencia Artificial han surgido varios modelos de Sistemas Inteligentes para soportar el desarrollo de Proyectos de Ingeniería de *Software*, desarrollados con una arquitectura de Sistemas Expertos Basados en Conocimiento, los cuales tienen como su nombre lo indica una Base de conocimientos construida con estructuras inteligentes y reglas de inferencia para deducir nuevos conocimientos a partir de los almacenados. Varios de los Sistemas Inteligentes para Ingeniería de *Software* se apoyan en herramientas CASE tradicionales, como lo demuestran los siguientes modelos:

### II.3.1 Perspectivas en la automatización de las Metodologías de Ingeniería de *Software*

A partir de un conjunto de perspectivas modernas de metodologías, aplicación de tecnologías de inteligencia artificial y programación orientada a objetos, Sergio Chapa Vergara y Pedro Alday Echevarría [SCP93] establecen la posibilidad de integrarlas totalmente en un ambiente automatizado.

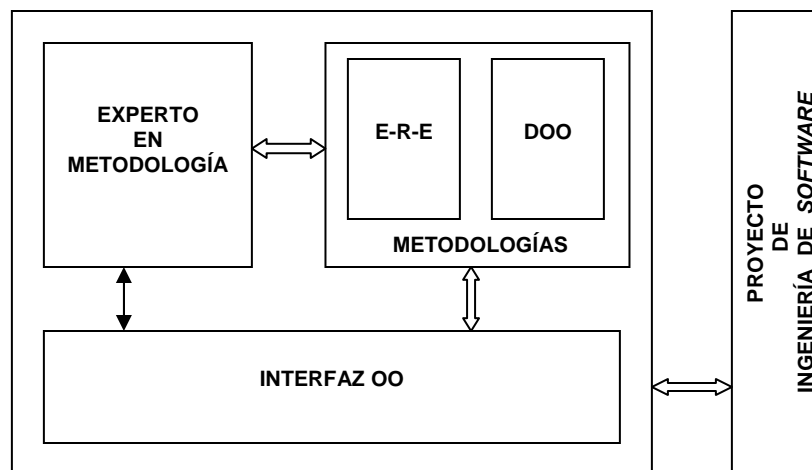


Figura II.3.- Modelo de SE con Metodologías E-R-E y DOO

Este *software* es concebido como un modelo semi-cerrado aplicando el método Entidad – Relación – Extendido (E-R-E) con un diseño y una interfaz Orientada a Objetos, como lo muestra la Figura II.3.

En un primer modelo, el Diseño Orientado a Objetos (DOO) reconocerá las entidades del modelo E-R-E como objetos. El modelo generará los diagramas específicos del DOO y el diagrama E-R-E.

Un segundo modelo que se obtiene es al aplicar metodologías estructuradas de Análisis y Diseño junto con Flujos de Control para especificar procedimientos detallados. El modelo produce diagramas de Flujo de Datos, Cartas Estructuradas y Diagramas de Flujo.

Un tercer modelo sustentado en Bases de Datos emplea un modelado semántico E-R-E complementado con un lenguaje de programación visual para crear un ambiente de consulta a la Base de Datos (BD). El modelo E-R-E incorpora un normalizador que produce un diseño físico de la BD en la Tercera Forma Normalizada 3FN. El lenguaje visual proporciona iconos predefinidos que el usuario selecciona, distribuye y conecta dentro de un espacio de edición y obtiene un programa visual. El cual se puede ejecutar o guardar para su posterior modificación o ejecución.

Este tercer modelo se desarrolla apoyándose de los trabajos realizados en el CINVESTAV; de BD de Fernando Fiorentino, Sistemas de Graficación Oscar Trejo, un modelador de diagramas E-R-E de Raúl Hernández y un lenguaje de Iconos para el desarrollo de aplicaciones de Sergio Chapa Vergara.

### II.3.2. MeRCI Sistema Experto para Reingeniería de *Software*

**MeRCI**, es el acrónimo en francés de *Método para Ingeniería Inversa Inteligente* desarrollado por Jacky Akoka, 1998 [JAA98], de la Ecole Supérieure de Sciences Economiques et Commerciales de Francia. La ingeniería inversa o reingeniería de *software* se refiere al proceso de descubrir como un sistema de *software* trabaja. La reingeniería de Base de Datos consiste en extraer esquemas conceptuales de las bases de datos (jerárquicas, red, relacionales u orientadas a objetos).

La reingeniería de *software* contribuye a tres principales factores (Waters & Chikofsky, 1994 [W&C94]):

- Identificar y entender los componentes y estructuras de sistemas de *software* existentes y sus relaciones internas.
- Descubrir y explicar como trabajan los componentes del sistema.
- Proveer una descripción de alto nivel de los componentes del sistema.

MeRCI es un sistema experto para reingeniería de *software* basado en el ciclo de vida de modelos desarrollados en sistemas de Base de Datos considerando cuatro estados en un orden inverso, de acuerdo al siguiente proceso:

- a) Extracción del esquema físico.- Este paso es para obtener una descripción completa del esquema físico. Este resultado puede ser seleccionado usando el diccionario de datos, DDL(Data Dictionary Lenguaje), las vistas, los sub-esquemas y analizando detalladamente el contenido de las salidas y entradas de pantallas de usuario. Puede ser necesario analizar el código fuente, con objeto de identificar algunas estructuras ocultas, estructuras no declaradas, y algunas especificaciones.
- b) Optimización del esquema físico.- A partir del esquema físico obtenido, se aplica un conjunto de reglas de inferencia de reingeniería para optimizarlo (Comyn-Wattiau & Akoka, 1996 [CWA96]). Con objeto de detectar la potencialidad de optimización se analizan las especificaciones del DDL, el DML(Data Management Lenguaje) y finalmente los datos.
- c) Conceptualización del esquema lógico.- Derivado de la extracción del esquema físico se aplica un conjunto de reglas de conocimiento para obtener el esquema lógico. Un siguiente conjunto de reglas permite obtener un modelo Entidad-Relación. El proceso de identificación de entidades, relaciones, cardinalidad y generalización, se ejecuta utilizando toda la información extraída en la etapa de optimización del esquema físico y se examina el código fuente de las declarativas en SQL para determinar las llaves primarias, columnas únicas, índices únicos y posibles candidatos a llaves.
- d) Descripción Semántica.- Iniciando desde el esquema conceptual, usando algunas hipótesis formuladas en la búsqueda de entidades, relaciones y generalizaciones, se caracterizan los objetos. El principal resultado obtenido es una formulación de un discurso con los requerimientos especificados por el usuario y la estructura de construcción del sistema usado en la descripción con la cual el sistema originalmente fue diseñado y desarrollado.

MeRCI ha sido aplicado en ejemplos complejos con bases de datos relacionales, lo que ha permitido incrementar su eficiencia.

### II.3.3. Herramienta ICASE para diseño reusando conocimientos adquiridos

El propósito de este sistema es facilitar el diseño automático de *software* en algún campo de la programación. Este sistema fue desarrollado por Hui Chen y Zenya Koono de Saitma University, Japón, y Nagayasu Tsutsumi de Hitachi Chubu *Software*, 1998 [CHM98] y [CHS98]. Una herramienta CASE(Computer Aided *Software* Engineering) convencional puede ser mejorada como un editor gráfico avanzado dedicado para diseño de *software* siguiendo algún método en Ingeniería de *Software*. Este ICASE (CASE Inteligente) diseña automáticamente *software*, reusando el conocimiento adquirido de diseño desde el CASE.

Las principales características de este sistema son las siguientes:

- El diseño automático de *software* similar al de un experto humano, basado en el conocimiento de diseños previos adquiridos automáticamente desde documentos de diseño.

- Uso de una herramienta estructurada CASE, para realizar el esquema inicial de diseño que servirá de monitor de los resultados obtenidos en el diseño automático.

Cuando un conjunto de conocimientos sistematizados es adquirido, y un modelo que hereda la estructura de conocimiento original es reforzado, la Ingeniería de *Software* prácticamente hace posible la reconstrucción del conocimiento dentro de un sistema experto de una forma automática [CH96a] y [CHN97].

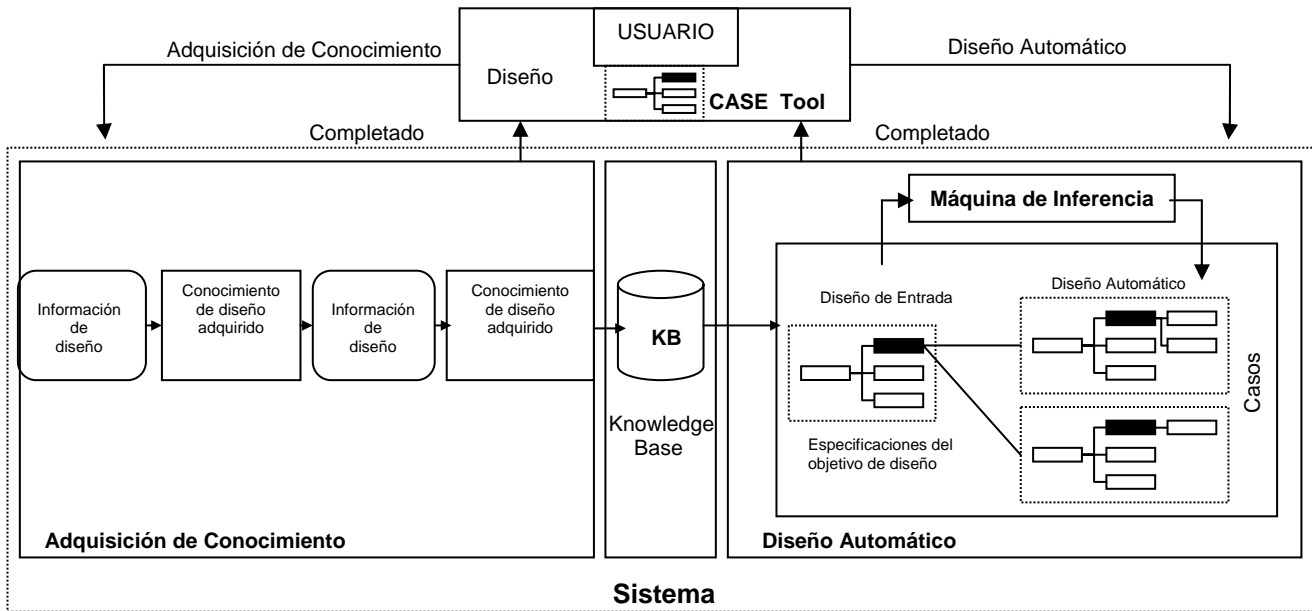


Figura II.4.- Estructura de una herramienta CASE Inteligente

La estructura de esta herramienta ICASE se muestra en la Figura II.4, está compuesta de una estructura de herramienta CASE modificada, una unidad de Adquisición de Conocimiento, una Base de Conocimientos y una unidad de Diseño Automático.

Un diseñador dibuja el prototipo del diseño usando símbolos gráficos con la herramienta CASE. El prototipo se diseña bajo especificaciones del usuario, considerando las funciones estandarizadas. Después el sistema es diseñado por equipos de funciones jerárquicas, cada especialidad en su respectivo campo de diseño, formando así lo que se denomina *Modelo Experto*.

Durante el diseño, el nivel alto de documentación es revisado por diseñadores de alto nivel y los niveles de detalle son revisados por grupos de trabajo. El sistema deja un proceso de trabajo jerárquico intersectado por documentos jerárquicos llamado *Modelo de Conocimientos*.

Este modelo es alimentado al Sistema Experto, y cuando los diagramas de diseño son llevados a pequeños pasos de diseño los conceptos en los documentos de entrada son transformados a documentos de salida. Sí el diseño es ejecutado por un Lenguaje Natural (LN) estandarizado en una forma uniforme, la expansión jerárquica llega a ser una pieza elemental reusable del conocimiento del proceso del diseño, llamada *regla de diseño*. Esto consiste de diagramas de símbolos con declarativas de LN.

Conectando estas reglas de diseño, el conocimiento del proceso de diseño forma una amplia jerarquía expandiendo una red semántica tanto como es posible. La forma convencional de generar una regla de diseño es a partir de una necesidad básica o de un conocimiento fundamental. El uso de reglas de diseño como una habilidad del nivel de reproducción hace que el sistema simule un diseñador experto mejor que al hacerlo con reglas básicas. El nivel superior controla el diseño por medio de un diagrama de estados de transición. El diseño final es evaluado para verificar que la información este completa, pronosticando las dificultades que pudieran ocurrir.

Los conocimientos pueden ser sistemáticamente adquiridos de los diseñadores expertos especificando detalles de la situación de acuerdo con las especificaciones del medio. De esta forma los conocimientos de diseño pueden ser adquiridos y reproducidos de manera similar a la forma original.

Una vez que el conocimiento es adquirido sistemáticamente el proceso de trabajo de diseño original jerárquico es tomado como un modelo de sistema experto que puede ser implementado. El punto de implementación [CH96a] es seguido por prácticas de Ingeniería de Software que también reflejan la estructura de conocimiento del diseñador experto. Agregando unidades expertas a la jerarquía se construye el Sistema Experto, incrementando su capacidad de automatismo y simulación del experto humano.

Con objeto de hacer más estándar la herramienta ICASE, realiza en el nivel más alto el detalle de los documentos.

### **III.3.4 Comparación de los Sistemas Inteligentes para Ingeniería de Software**

El conjunto de Sistemas Inteligentes descritos en los apartados anteriores de este Capítulo, describen diversos enfoques que sirven como herramientas para Ingeniería de Software. Cada uno con Arquitectura Basada en Conocimientos, que mediante procesos inteligentes apoyan a los especialistas de software a incrementar su productividad, calidad y eficiencia.

La arquitectura deseable para un Sistema Inteligente Integral para Ingeniería de Software, es incorporar una Base de Conocimientos genérica de clases de marcos, que integre los diferentes niveles de Ingeniería de Software, sin embargo, por lo extenso que es este tema del conocimiento de la Ingeniería, no es posible abarcar de forma específica todos los niveles de conocimiento. Los Sistemas Inteligentes presentados, son de una arquitectura específica que resuelven de forma concreta problemas de BD y de diseño de modelos de sistemas. Estas diferencias más que ser una brecha entre Sistemas Inteligentes de Ingeniería de Software, motiva la realización de un trabajo futuro para la integración entre las diferentes herramientas presentadas en este Capítulo.

Analizando las características de la arquitectura de un Sistema Inteligente como "Herramienta Integral de Ingeniería de Software" con respecto a la arquitectura de los diferentes Sistemas Expertos descritos en este Capítulo; Automatización de Metodologías de Ingeniería de Software, MeRCI e ICASE para diseño de software, se realizó una tabla (Figura II.5), en donde se establece el mapa del dominio particular de cada solución, observando que es posible realizar interfaces para construir una Herramienta Integral.

<b>Arquitectura</b>	<b>Integral</b>	<b>Perspectivas de Automatización</b>	<b>MeRCI</b>	<b>ICASE</b>
Knowledge Base	✓	✓	✓	✓
Adquisición de Conocimiento	✓			✓
Lenguaje Natural	✓			✓
Generación Automática De Proyectos	✓	✓		
Integración de Metodologías	✓	✓		
Modelos de BDD	✓	✓	✓	✓
Métricas y Estadísticas	✓			
Diseño de Procesos (Métodos)	✓	✓	✓	✓
Reingeniería de Procesos	✓		✓	
Integración con CASE	✓			✓
Integración con <i>software</i> para Administración de Proyectos	✓			

Figura II.5.- Cuadro comparativo de Sistemas Inteligentes de Ingeniería de *Software*

En esta forma el Estado del Arte de las diversas arquitecturas de tecnologías que apoyan a la Ingeniería de *Software* nos muestra que es posible desarrollar mediante la integración de las diferentes herramientas un Sistema Inteligente para la Ingeniería de *Software* que mejore a las actuales herramientas CASE en el desarrollo de Sistemas de Información.

Por el momento al finalizar este Capítulo se tiene la confianza de que el Modelo del Sistema Inteligente Integral partiendo de un PROTOTIPO para la Administración de Proyectos de Ingeniería de *Software* y sus metodologías se encuentran en situación aceptable de desarrollo sin duplicar productos existentes en el mercado.

## Capítulo III. Dominio del conocimiento de la HIS para el PROTOTIPO SE-APIS “Ingeniería de Software”

El dominio del conocimiento del PROTOTIPO del Sistema Basado en Conocimiento como Herramienta de apoyo para la Administración de Proyectos se encuentra en el ámbito de la Ingeniería de *Software*, dividido en tres subdominios o módulos, el primero son las Metodologías de desarrollo, el segundo la Administración de Proyectos [PMI96] y el tercero aunque de forma incipiente son las Métricas de desarrollo para la Ingeniería de *Software* [RPE90] y [RPE02].

En este capítulo se describen los conceptos fundamentales de cada uno de los subdominios, a partir de los cuales se obtienen los conocimientos para alimentar la base de conocimientos del sistema experto a desarrollar en este trabajo de tesis.

Cabe aclarar que en el PROTOTIPO del SE-APIS solo se incorpora las métricas para el Modelo de PUTNAM [PUT78] y [PUT92].

### Definición de Ingeniería de *Software*

Una primera definición de ingeniería de *software* fue propuesta por Fritz Bauer [NAU69] en la primera conferencia importante dedicada al tema:

*El establecimiento y uso de principios de ingeniería robustos, orientados a obtener económicamente software que sea fiable y funcione eficientemente sobre máquinas reales.*

Aunque se han propuesto muchas más definiciones globales, todas refuerzan la importancia de una disciplina de ingeniería para el desarrollo del *software*.

La ingeniería de *software* abarca un conjunto de tres elementos clave: métodos, herramientas y procedimientos, que facilitan el control del proceso del desarrollo de *software* y suministra a los que practican dicha ingeniería las bases para construir *software* de alta calidad de una forma productiva. En los párrafos que siguen, se examina brevemente cada uno de estos elementos.

Los métodos de la ingeniería de *software* suministran el “cómo” construir técnicamente el *software*. Los métodos abarcan un amplio espectro de tareas que incluyen: planificación y estimación de proyectos; análisis de los requerimientos del sistema y del *software*; diseño de estructuras de datos, arquitectura de programas y procedimientos algoritmos, codificación; prueba y mantenimiento. Los métodos de la ingeniería del *software* introducen frecuentemente una notación especial orientada a lenguaje o gráfica y un conjunto de criterios para la calidad del *software*.

Las herramientas de la ingeniería de *software* suministran un soporte automático o semiautomático para los métodos. Como se menciona en el Capítulo II, hoy existen



herramientas para soportar cada uno de los métodos. Cuando se integran las herramientas de forma que la información creada por una herramienta pueda ser usada por otra, se establece un sistema para el soporte del desarrollo del *software*, llamado ingeniería del *software* asistido por computadora (CASE: acrónimo en inglés de “computer-aided *software* engineering”). CASE combina el *software*, hardware y bases de datos de la ingeniería del *software* [HEN84].

Los procedimientos de la ingeniería de *software* son el enlace que pega a los métodos y herramientas y facilita un desarrollo racional y oportuno del *software* de computadora. Los procedimientos definen la secuencia en la que se aplican los métodos, las entregas (documentos, informes, formas, etc.) que se requieren, los controles que ayudan a asegurar la calidad y coordinar los cambios, y las guías que facilitan a los gestores del *software* establecer su desarrollo.

### III.1. Módulo de Conocimiento de Metodologías de Desarrollo

La ingeniería de *software* está compuesta de pasos que abarcan los métodos, herramientas y procedimientos, estos pasos se denominan frecuentemente paradigmas de la ingeniería del *software*, y forman lo que se conocen como Metodologías de Desarrollo. Un paradigma para la ingeniería de *software* se elige basándose en la naturaleza del proyecto y de la aplicación, los métodos, herramientas a usar y los controles y entregables requeridos. En el desarrollo de sistemas se han tratado ampliamente varios modelos de metodologías que se describen en las siguientes secciones.

#### III.1.1. Modelos del ciclo de vida de un sistema.

En el mundo de la ingeniería de *software*, se utiliza la expresión ciclo de vida para describir las principales fases en la vida de un *software*, sistema o metodología. El ciclo de vida en general se puede ver bajo los siguientes tres modelos:

**Secuencial:** Una vez que se ha terminado uno de los pasos nunca más se regresa a éste o a cualquier otro paso previo a ese paso. Es recomendable para proyectos que no son críticos o que son muy pequeños.

**Iterativo:** Desde el punto de vista de este enfoque, si hay suficiente razón para hacer algo, se hace; es decir que uno puede regresar a un paso anterior aunque ya haya sido terminado e introducir algún cambio y luego propagar los efectos de este cambio a los pasos siguientes.

**Recursivo:** Permite la repetición o reaplicación de las mismas acciones una vez que se ha terminado el producto. Todos los métodos recursivos son iterativos, pero todos los métodos iterativos no son recursivos.

En el modelo **secuencial de cascada**, se da por hecho que todo está correcto a la primera vez ya que cada paso debe quedar completamente terminado antes de pasar al siguiente, y además se aplica la regla de no regresar ni saltar a pasos siguientes. Se requiere que todos

los requerimientos del análisis sean terminados antes de comenzar el diseño y que todo el diseño sea terminado antes de empezar el código.

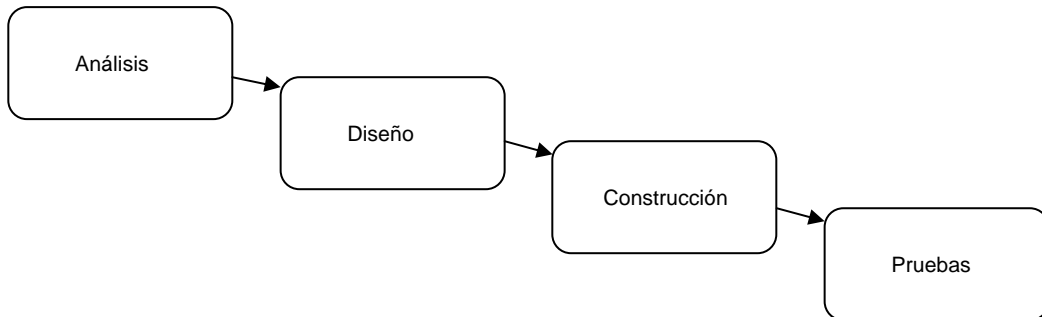


Figura III.1.- Modelo secuencial de cascada

El modelo **iterativo en cascada**, es el más común. En esta metodología, se requiere que se termine completamente un paso, se verifiquen sus resultados y luego continúe con el paso siguiente. Se puede retornar en cualquier momento a un paso anterior ya sea para terminarlo, para hacer algún cambio y propagarlo en los pasos que el cambio realizado afecte.

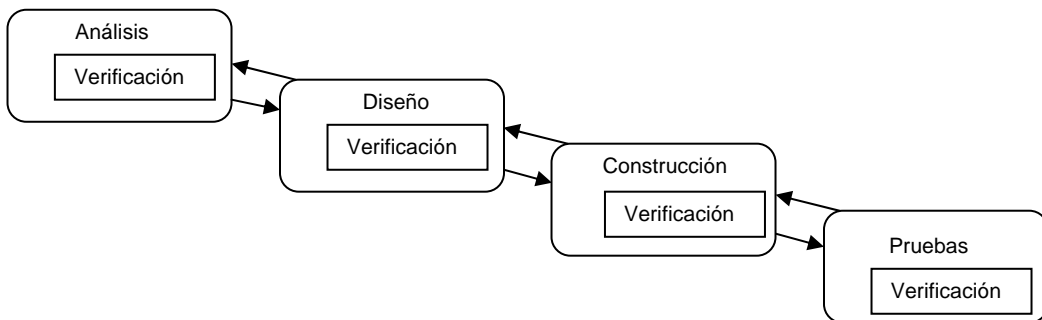


Figura III.2.- Modelo iterativo de cascada

El **“Modelo b”**, se reconoce la fase de mantenimiento en el ciclo de vida del *software*. Entendiendo que el *software* permanece por un período de tiempo, el mantenimiento casi siempre involucra un mejoramiento, por lo que esta fase de mantenimiento es muy parecida a la fase de desarrollo del *software*, por lo que en su representación gráfica se asemeja a una b minúscula, presentado en la Figura III.3.

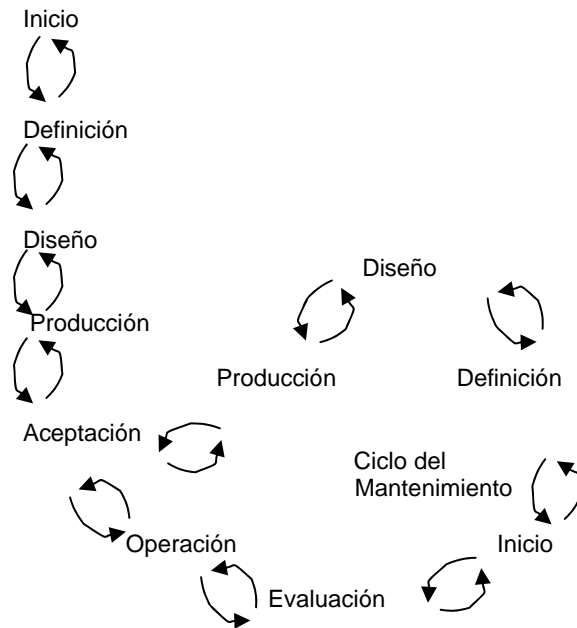


Figura III.3.- Modelo "b"

El ciclo de vida de acuerdo al "**Modelo de Espiral**" de Boehm, [BOE88] y [BOE98], es un modelo iterativo e interactivo mediante el cual un sistema es desarrollado en una serie de versiones incrementales. Un proyecto de *software* puede ser desarrollado con un ciclo de vida con un modelo de espiral con cuatro ciclos y cuatro cuadrantes como se ilustra en la Figura III.4.

- Ciclo de prototipo conceptual – captura los requerimientos del negocio, define metas para prototipo conceptual, produce el diseño conceptual del sistema, diseña y construye el prototipo, produce y acepta el plan de pruebas, conduce el análisis de riesgo y hace recomendaciones.
- Primer ciclo de construcción – deriva los requerimientos del sistema, define metas para la primera construcción, produce el diseño lógico del sistema, diseña y produce la primera construcción, produce planes de prueba del sistema, evalúa la primera construcción y hace recomendaciones.
- Segundo ciclo de construcción – deriva los requerimientos de subsistemas, define metas para la segunda construcción, produce el diseño físico, realiza la segunda construcción, produce planes de prueba del sistema, evalúa la segunda construcción y hace recomendaciones.
- Ciclo final – completa los requerimientos, diseño final, construcción final, ejecuta pruebas unitarias de subsistemas e integrales del sistema y acepta las pruebas.

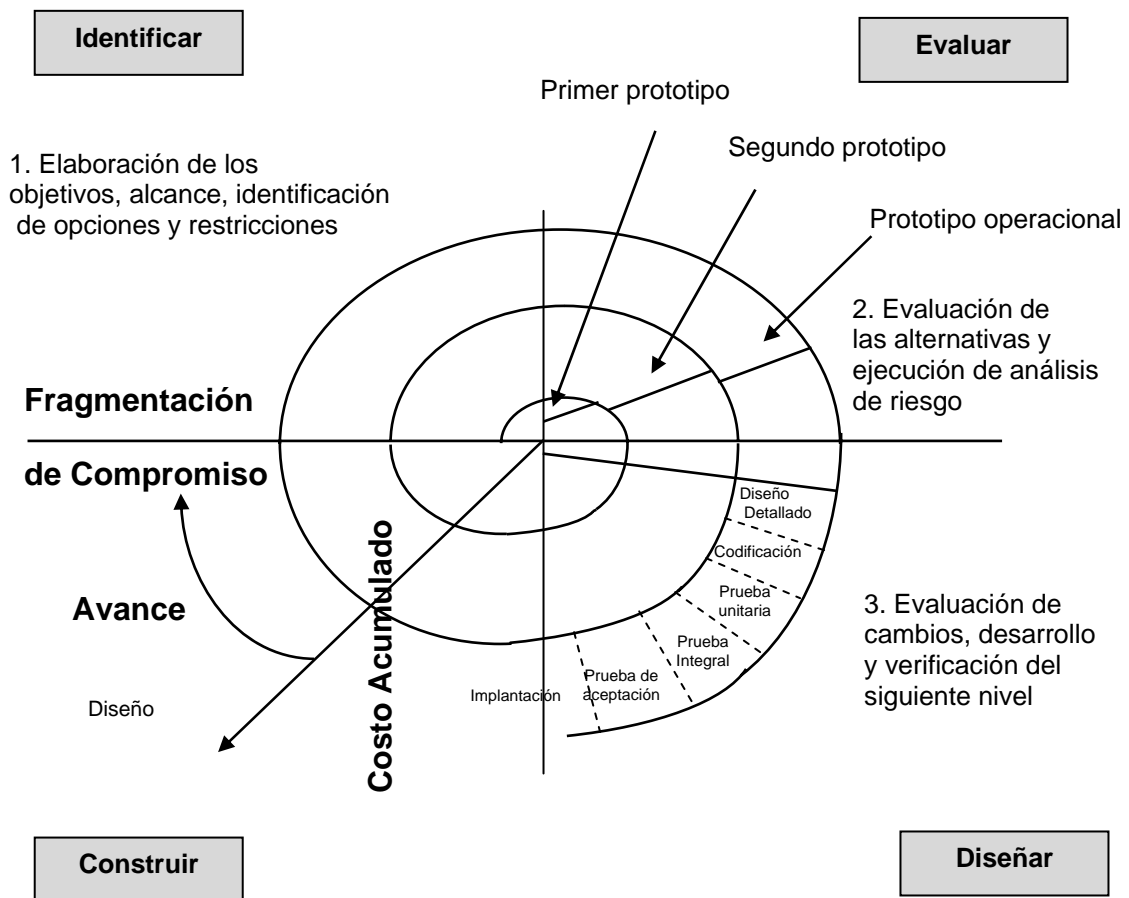


Figura III.4.- Modelo de espiral

A medida que crece la espiral, el costo global del producto lo determina el radio de la espiral y el avance lo determina el desplazamiento angular. La espiral se desplaza del centro hacia fuera. Las tres observaciones importantes a este modelo de espiral, son:

1. Este modelo se recomienda para proyectos de investigación en los cuales los requerimientos, alternativas y restricciones, son desconocidas a su inicio.
2. Presenta un modelo de organización para la construcción de prototipos.
3. Aunque este modelo de espiral involucra la construcción de prototipos, el modelo de construcción de prototipos no constituye un modelo de espiral.

### III.1.2. Ciclo de vida clásico

La Figura III.5 ilustra el paradigma del ciclo de vida clásico para la ingeniería del *software*, el cual es un “**modelo de cascada**”, este paradigma exige un enfoque sistemático, secuencial del desarrollo de *software*.

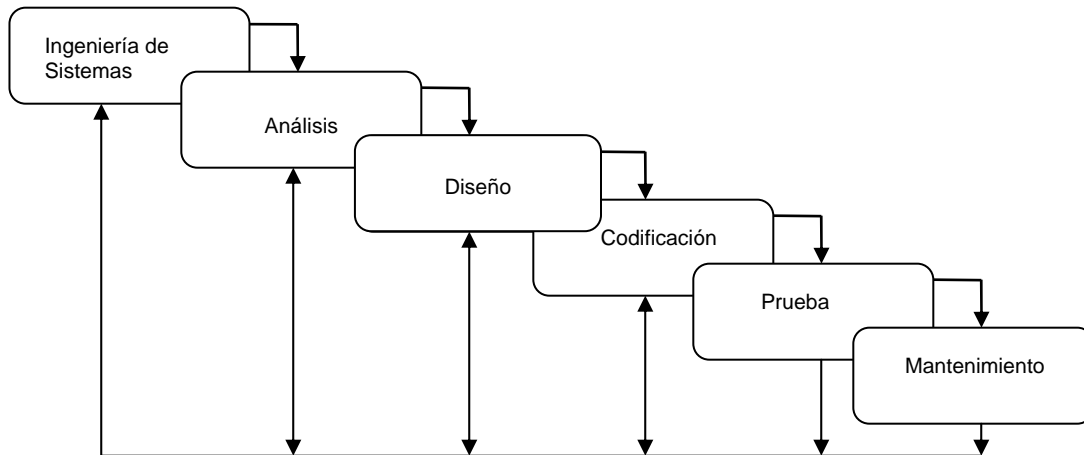


Figura III.5.- El ciclo de vida clásico

El ciclo de vida clásico es la metodología más ampliamente usada en la ingeniería del *software*. Sin embargo, con el paso de los años, se han producido críticas al paradigma. Entre los problemas que se presentan cuando se aplica el paradigma del ciclo de vida clásico se encuentran:

1. Los proyectos reales raramente siguen el flujo secuencial que propone el modelo. Siempre ocurren iteraciones y se crean problemas en la aplicación del paradigma.
2. Normalmente es difícil para el cliente establecer explícitamente al principio todos los requerimientos. El ciclo de vida clásico requiere esto y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos proyectos.
3. El cliente debe tener paciencia. Una versión funcionando del programa no estará disponible hasta las etapas finales del desarrollo del proyecto. Un error importante no detectado hasta que el programa esté funcionando puede ser desastroso.

Cada uno de estos problemas es real. Sin embargo, el paradigma clásico del ciclo de vida tiene un lugar definido e importante en el trabajo sobre ingeniería de *software*. Suministra un patrón en el que puede colocarse los métodos para el análisis, diseño, codificación, prueba y mantenimiento. Además, los pasos del paradigma clásico del ciclo de vida son muy similares a los pasos genéricos aplicables a todos los paradigmas de ingeniería de *software*. El ciclo de vida clásico permanece como el modelo procedimental más ampliamente usado por los ingenieros de *software*

### III.1.3. Construcción de prototipos

Se utiliza normalmente cuando un usuario define un conjunto de objetivos generales para el *software*, pero no identifica los requerimientos detallados de entrada, procesamiento o salida. En otros casos el programador puede no estar seguro de la eficiencia de un algoritmo, la adaptabilidad de un sistema operativo o la forma en que debe realizarse la interacción hombre-máquina.

La construcción del prototipo es un proceso que facilita al programador la creación de un modelo del *software* a construir. El modelo tomará una de las tres formas siguientes: un “prototipo en papel” que describa la interacción hombre-máquina de forma que facilite al usuario la comprensión de cómo se producirá tal interacción, un “prototipo que funcione” que implementa algunos subconjuntos de la función requerida del *software* deseado, o un programa existente que ejecute parte o toda la función deseada, pero que contenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo.

La secuencia de los sucesos para el paradigma de construcción de prototipos se muestra en la Figura III.6. El técnico y el usuario se reúnen y definen los objetivos globales para el *software*, identifican todos los requerimientos conocidos y perfilan las áreas en donde será necesaria una mayor definición, luego se produce un “diseño rápido”.

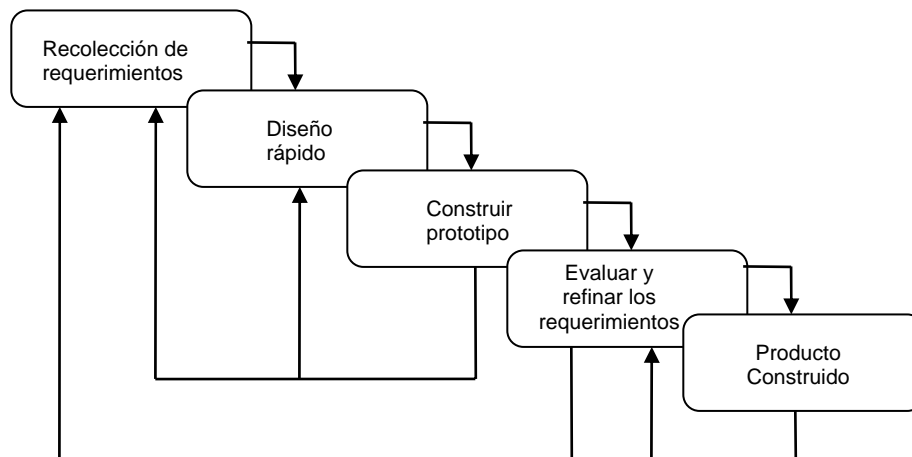


Figura III.6.- Construcción de prototipos

El diseño rápido se enfoca sobre la representación de los aspectos del *software*, visibles al usuario (por ejemplo, métodos de entrada y formatos de salida). El diseño rápido conduce a la construcción de un prototipo. El prototipo es evaluado por el cliente/usuario y se utiliza para refinar los requerimientos del *software* a desarrollar. Se produce un proceso interactivo en el que el prototipo es “afinado” para que satisfaga las necesidades del cliente, al mismo tiempo que facilita al que lo desarrolla una mejor comprensión de lo que hay que hacer. Idealmente, el prototipo sirve como un mecanismo para identificar los requerimientos del *software*.

La construcción de prototipos como paradigma para la ingeniería de *software*, puede ser problemático [BRO75] por las siguientes razones:

1. El cliente ve funcionando lo que parece ser una versión del *software*, ignorando que el prototipo se ha hecho con “chicle y alambres”.
2. Puede utilizarse un sistema operativo o lenguaje de programación inapropiado simplemente porque está disponible y es conocido, un algoritmo ineficiente puede implementarse de forma sencilla para demostrar su capacidad.

Aunque se pueden presentar problemas, la construcción de prototipos es un paradigma efectivo para la ingeniería de *software*. La clave está en definir al comienzo las reglas del juego, esto es, el cliente y técnico deben estar de acuerdo en que el prototipo se construya para servir sólo como un mecanismo de definición de los requerimientos.

#### III.1.4. Modelo Orientado a Objetos

Un nuevo enfoque para construir *software* que deje atrás los métodos de la programación convencional y ofrezca una mejor forma de construir sistemas pequeños y de gran escala que sean confiables, flexibles, mantenibles y capaces de evolucionar para satisfacer los requerimientos del cambio, es la Tecnología de *Programación Orientada a Objetos* [POO98].

Las técnicas orientadas a objetos permiten que el *software* se construya a partir de *objetos* de comportamiento específico. Los propios objetos se pueden construir a partir de otros que a su vez pueden estar formados por otros objetos. Esto nos recuerda una maquinaria compleja construida por partes, subpartes, sub - subpartes, etc.

Las etapas a seguir en esta metodología son las siguientes:

1. Análisis.- Comienza con el planteamiento del problema, se construye un modelo que exprese las características principales del problema real que se va a resolver. El modelo de Análisis es una abstracción precisa y concisa del *qué* debe hacer el sistema y no del *cómo* lo hará.
2. Diseño.- El diseñador de sistemas toma decisiones de alto nivel a cerca de la arquitectura global del sistema durante el diseño del sistema.
3. Diseño de Objetos.- El diseñador de objetos construye un modelo con base en el modelo de análisis pero incluyendo los detalles de la implementación. El diseñador agrega los detalles al modelo de diseño de acuerdo con la estrategia establecida durante el diseño del sistema. El foco del diseño del objeto es la estructura de datos y el algoritmo para implementar cada clase.
4. Implementación.- Las clases objeto y la relación entre ellas desarrolladas durante el diseño de los objetos son finalmente trasladadas a un lenguaje de programación particular, una base de datos o una implementación en *hardware*.

La programación es relativamente la menor parte del ciclo de vida del desarrollo del sistema teniendo en cuenta que las decisiones fuertes fueron tomadas en la etapa del diseño.

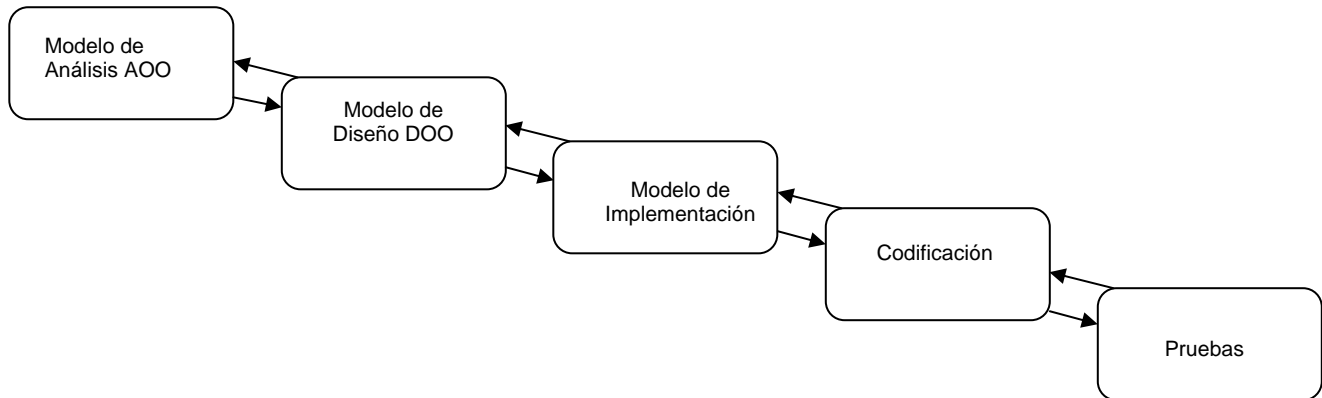


Figura III.7.- Ciclo de vida Orientada a Objetos

El concepto de orientado a objetos puede ser aplicado en todo el ciclo de vida del desarrollo del sistema, desde el análisis, diseño e implementación.

Los conceptos de identidad, clasificación, *poliformismo* y herencia manejados en orientada a objetos, se aplican a través del ciclo completo del desarrollo del sistema.

No todas las clases de un sistema forman parte del análisis sino que pueden ser introducidas en el diseño o en la implementación.



## III.2. Módulo de Conocimiento de Administración de Proyectos de *Software*

Un proyecto de *software* es un compromiso temporal para crear un producto o servicio único. Temporal significa que cualquier proyecto tiene un inicio y un fin definidos. Único significa que el producto es diferente en algo que lo distingue de otros productos o servicios similares. Los proyectos comparten varias características:

- Son ejecutados por personas.
- Restringidos por recursos limitados.
- Planeados, ejecutados y controlados.

La administración de proyectos de *software* es la aplicación de conocimiento, habilidades, herramientas y técnicas para proyectar actividades con objeto de encontrar o cubrir los requerimientos necesarios y expectativas. Encontrar o cubrir las expectativas invariablemente involucran el equilibrio siempre en conflicto entre:

- Alcance, tiempo, costo y calidad.
- Mantener la diferencia entre necesidades y expectativas.
- Identificar requerimientos y expectativas no bien definidas.

### III.2.1. Fases y ciclo de vida de un proyecto de *software*.

Debido a que los proyectos de *software* son únicos, involucran un grado de incertidumbre. Las organizaciones que realizan proyectos usualmente dividen cada proyecto en varias *fases* para poder administrar y controlar adecuadamente, y establecer los enlaces apropiados para la ejecución de las operaciones que la organización ejecutará. De manera colectiva las *fases del proyecto* son llamadas *ciclo de vida del proyecto de software*.

#### Características de las fases de un proyecto.

Para cada fase del proyecto se define su alcance para ser completado en uno o más entregables. Un entregable es un tangible y verificable producto, por ejemplo un estudio de factibilidad, un diseño detallado o un prototipo operando. Los entregables y las fases definidas son parte generalmente de un diseño lógico secuencial para asegurar la definición adecuada de un producto del proyecto.

La conclusión de una fase de un proyecto está generalmente marcada por una revisión de la lista de entregables y del comportamiento del proyecto con objeto de: a) determinar si el proyecto debe continuar para la siguiente fase, b) detectar y corregir efectivamente errores de costo. Estos fines de fase revisados son frecuentemente llamados *fin de fase, cambio de estado o puntos de cancelación*.

Cada fase del proyecto normalmente incluye un conjunto de productos de trabajo diseñados para establecer el nivel deseado de administración de control. La mayoría de estos conceptos están relacionados con la primera fase de entregables, y las fases típicas toman

su nombre de los conceptos: **planeación, análisis, diseño, construcción, prueba, implantación, mantenimiento**, y otros apropiados de acuerdo al proyecto de *software*.

### **Características del ciclo de vida del proyecto de *software*.**

El ciclo de vida del proyecto sirve para definir el inicio y el final de un proyecto. En la definición del ciclo de vida del proyecto se podría determinar que el estudio de factibilidad sea tratado como la primer fase del proyecto, como una fase separada o como un proyecto independiente.

La definición del ciclo de vida del proyecto deberá también determinar las acciones transicionales a incluir en el fin del proyecto. De esta manera, el ciclo puede ser usado para ligar el proyecto a las operaciones realizadas en la organización.

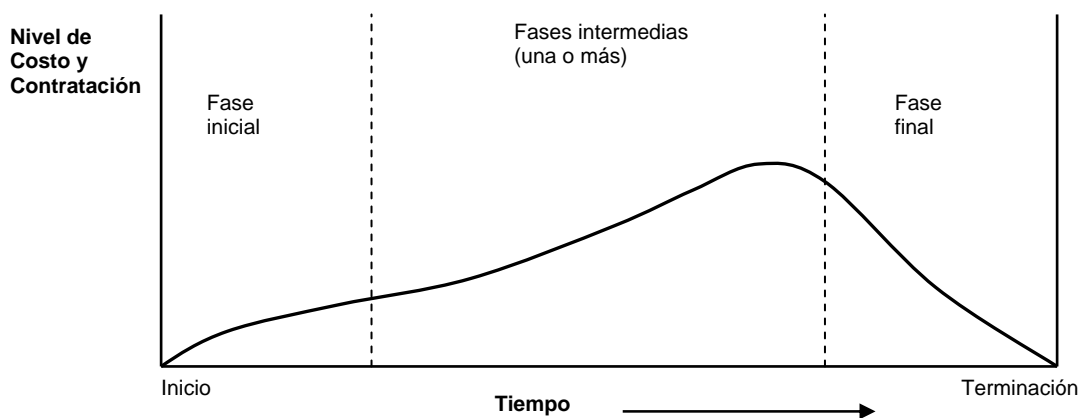


Figura III.8.- Ciclo de vida típico de un proyecto de *software*

El ciclo de vida del proyecto generalmente define:

- El trabajo técnico que debe hacerse en cada fase.
- Quién participa en cada fase.

De acuerdo con el PMBOOK [PMI96], la descripción del ciclo de vida del proyecto comparte un número común de características:

- El costo y nivel de personal son menores en el inicio, para elevarse en la parte intermedia y bajar rápidamente cuando el proyecto llegue a su conclusión (Figura III.8).
- La probabilidad de completar exitosamente el proyecto es más baja y el riesgo y la incertidumbre son más altas. La probabilidad de éxito se incrementa progresivamente conforme el proyecto continúa.
- La habilidad de administrar para influir en las características finales del producto del proyecto y su costo final es más alta en el inicio y baja progresivamente conforme el proyecto avanza. Una mayor contribución para este fenómeno es que el costo de cambio de alcance y errores generalmente se incrementan conforme el proyecto continúa.

## Curva típica de Avance de un proyecto de Ingeniería de *software*

La línea base planeada (Morris, Peter W.G. 1981 [PMI96]) es ilustrada en la Figura III.9, su comportamiento puede generalizarse en las fases siguientes:

- Factibilidad – es la formulación del proyecto, estudio de factibilidad, y aprobación.
- Planeación y preanálisis - costo y programación, términos contractuales y condiciones, planeación detallada. Los mayores contratos son hechos al final de esta fase.
- Desarrollo – análisis, diseño, construcción, instalación y pruebas. La implantación es substancialmente completada al finalizar esta fase.
- Liberación y puesta en marcha – prueba final y mantenimiento. La implantación es puesta en operación completa al final de esta fase.

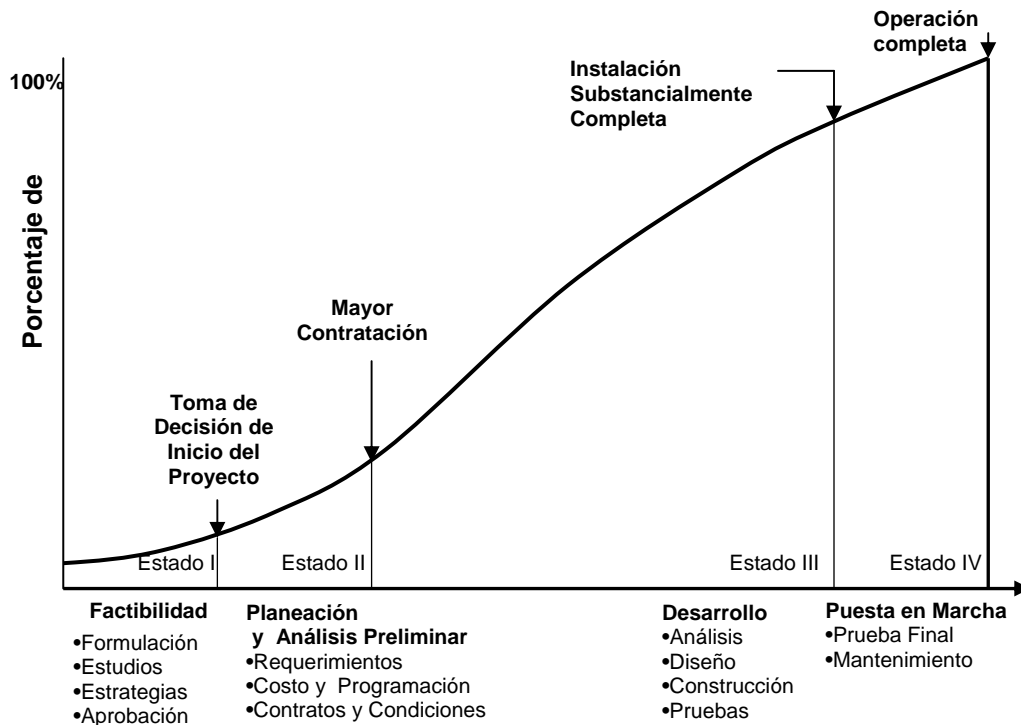


Figura III.9.- Ciclo típico de un proyecto de *software*

### III.2.2. WBS, definición del alcance de un proyecto de *software*.

La administración del alcance del proyecto de *software* incluye los procesos suficientes para asegurar que el proyecto incorpora todo el trabajo requerido, y únicamente el trabajo requerido para completar el proyecto con éxito.

La definición del alcance de un proyecto incluye la subdivisión de los principales entregables del proyecto dentro de otros más pequeños y manejables componentes con objeto de:

- Mejorar la exactitud de costo, tiempo y recursos estimados.
- Definir una línea base para ejecutar mediciones y control.
- Facilitar claramente la asignación de responsabilidades.

Dentro de las técnicas y herramientas que apoyan la definición del alcance y estructura del proyecto esta el WBS.

## WBS – Work Breakdown Structure

El WBS de un proyecto puede ser usado como un *template* para un proyecto nuevo. Aunque cada proyecto es único, WBS puede ser reutilizado puesto que la mayoría de los proyectos tienden a semejarse entre sí. Por lo general los proyectos de una organización dada deben tener el mismo o similar ciclo de vida, así como entregables similares requeridos para cada fase. Un ejemplo de WBS se muestra en la Figura III.10.

Un WBS esta orientado a entregables agrupados de elementos de un proyecto, que organizan y definen el total del alcance del proyecto, el trabajo que no está en el WBS no está en el proyecto. El WBS es frecuentemente usado para desarrollar y confirmar un conocimiento común del alcance del proyecto. Cada nivel descendente representa un incremento en el detalle de la descripción de los elementos del proyecto.

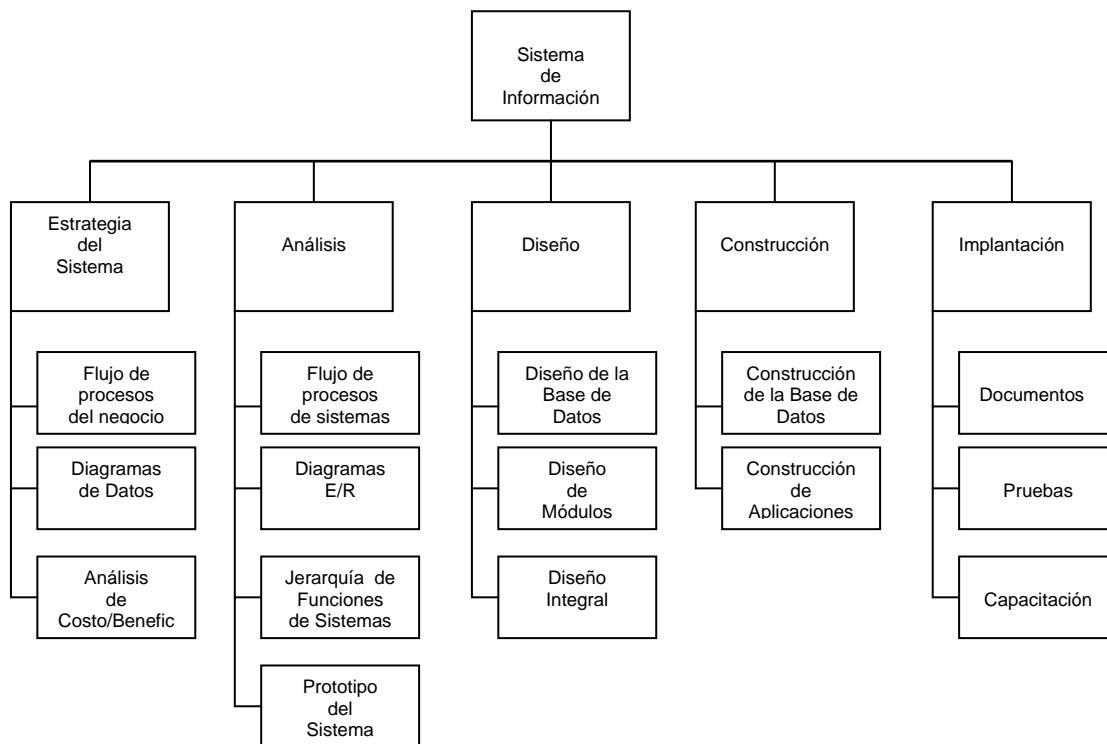


Figura III.10.- *Work Breakdown Structure* de un proyecto de *software*

Para cada elemento del WBS generalmente es asignado un identificador único; estos identificadores son comúnmente conocidos como *código de cuenta*. Los elementos del más bajo nivel del WBS son frecuentemente conocidos como *paquetes de trabajo*. Estos paquetes de trabajo pueden ser descompuestos más adelante en *actividades*.

Los elementos descritos son frecuentemente colectados en un *diccionario WBS*. Un diccionario WBS puede típicamente incluir la descripción de paquete de trabajo como otra información de planeación adicional a fechas programadas, presupuestos y asignación de contratos.

### III.2.3. Planeación del tiempo del proyecto de software

La planeación del tiempo de un proyecto incluye los procesos requeridos para asegurar en tiempo la ejecución completa del proyecto. Dentro de los procesos están: *Definición de actividades, Secuencias de actividades, Estimación de la duración, Administración del desarrollo y Administración del control, en este punto explicaremos algunas de ellas.*

#### III.2.3.1. Definición y secuenciación de actividades

La definición de una actividad incorpora la identificación y documentación de actividades específicas que deben ser ejecutadas con objeto de producir los entregables y subentregables identificados en el WBS. Implícitamente en este proceso es necesario definir las actividades tales que los objetivos del proyecto se cumplan.

Los resultados de la definición de actividades son: *la lista de actividades, el soporte detallado y la actualización del WBS.*

La lista de Actividades debe incluir todas las actividades que se ejecutarán en el proyecto, y deben organizarse como una extensión del WBS para ayudar a asegurar que está completo y que no incluye ninguna actividad no requerida como parte del alcance del proyecto. Al igual que el WBS, la lista de actividades debe incluir la descripción de cada actividad para asegurar a los miembros del equipo del proyecto como se realizará el trabajo.

El soporte detallado para la lista de actividades debe ser documentado y organizado como sea necesario para facilitar su uso en la administración de los procesos del proyecto. También debe incluir la documentación de todas las suposiciones y restricciones.

Usando el WBS para identificar las actividades necesarias, el equipo del proyecto puede identificar entregables olvidados o puede determinar cual descripción de entregable debe ser aclarada y corregida. Cualquier actualización debe ser reflejada en la documentación del WBS.

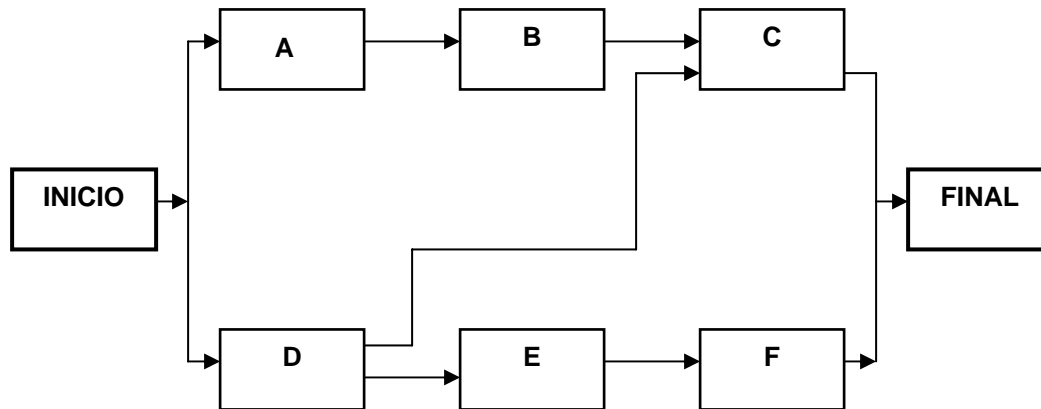
#### Secuenciación de actividades

La secuenciación de actividades involucra la identificación y documentación de las dependencias entre las actividades. Las actividades deben ser secuenciadas con exactitud.

El **método de diagramación de precedencias (PDM)** es un método de construcción de un diagrama de red del proyecto usando cajas para representar las actividades y conectándolas con flechas que muestren las dependencias. La Figura III.11 muestra un ejemplo sencillo usando PDM. Esta técnica es también llamada *caja-en-actividad* y es el método usado para la mayoría del *software* de administración de proyectos. Este incluye cuatro tipos de dependencias o relación de precedencias:

1. Fin-Inicio – la actividad debe terminar para que inicie la siguiente.
2. Fin-Fin – la actividad debe terminar para que termine la otra.

3. Inicio-Inicio – la actividad debe iniciar para que la otra pueda iniciar.
4. Inicio-Fin – la actividad debe iniciar para que la otra pueda terminar.



III.11.- Diagrama del Método de Modelo de Precedencias (PDM)

En PDM la relación Fin–Inicio es la comúnmente usada, Inicio-Fin raramente se usa. Usando Inicio-Inicio, Fin-Fin e Inicio-Fin en un *software* de administración de proyectos si no se ligan con duraciones que coincidan perfectamente en el tiempo, pueden obtenerse resultados inesperados puesto que estas relaciones no han sido consistentemente implementadas.

### III.2.3.2. Programación y Método de la ruta crítica del proyecto de *software*

La *estimación de la duración* incluye evaluaciones del número de periodos de trabajo que serán necesarios para completar cada una de las actividades identificadas. Estimar el número de periodos de trabajo requiere completar la información de una actividad con su tiempo de duración. Los miembros del equipo del proyecto deberán conocer ampliamente las actividades o al menos tener idea de lo que se debe realizar.

El desarrollo de la **programación de un proyecto** significa determinar las fechas de inicio y terminación de las actividades. Sí el inicio y terminación no son reales, el proyecto no terminará como fue programado. El proceso de la programación debe frecuentemente ser revisado

Las herramientas y técnicas para la programación de un proyecto son análisis matemáticos que calculan teóricamente las fechas más tempranas y más tardías para todas las actividades del proyecto sin estimar las limitaciones de recursos. Las fechas resultantes no son el programa, pero indican un rango del periodo de tiempo dentro del cual las actividades deben ser programadas dando limites de recursos y restricciones. Los análisis matemáticos mayormente conocidos en el desarrollo de *software* son es el **CPM** y el **PERT** [MOD83].

### Método de la ruta crítica (CPM)

Para planear, programar y controlar el desarrollo de un proyecto se divide en sus actividades de conformidad con el nivel de detalle requerido. Para llevar a cabo algunas actividades es necesario haber terminado otras. A éstas se les llama predecesoras de las primeras, a las cuales se les llama sucesoras.

A cada proyecto podemos asociarle un grafo formado por arcos dirigidos (o flechas) y nodos, el arco representa una actividad. A cada arco, se le asocia el tiempo para ejecutar la actividad correspondiente.

Los arcos dirigidos se dibujan de tal forma que un extremo del arco se apoya sobre las puntas de sus predecesores inmediatos. Así, en la Figura III.12 las actividades A, B, C son predecesoras inmediatas de la actividad D. En los extremos tienen nodos, donde se unen con otras actividades.

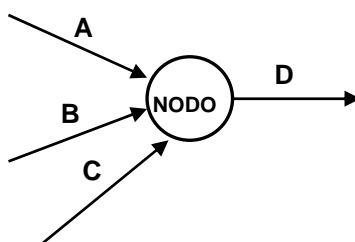


Figura III.12.- Relación entre actividades

En algunos casos las relaciones de predecesión son tales que se hace necesario introducir actividades “dummy” de duración y costo nulos. Esto sería el caso de un proyecto que tuviese la siguiente tabla de predecesión, cuya gráfica se muestra en la Figura III.13.

ACTIVIDAD	PREDECESORES INMEDIATOS
A	-----
B	-----
C	A
D	A,B

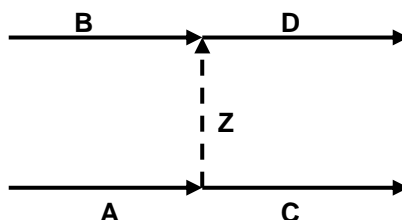


Figura III.13.- Gráfica de precedencias con una actividad *dummy*

En la gráfica, el predecesor inmediato de C es A y el de D es B. Sin embargo, se requiere que A también sea predecesor inmediato de D. Si los arcos dirigidos A y B terminasen en un

mismo punto, todos los sucesores inmediatos de A serían también sucesores inmediatos de B. Pero, de acuerdo con los datos, no obstante, que C es sucesor inmediato de A, no lo es de B. Este problema se resuelve introduciendo, como mostramos en la Figura III.13, una actividad muda Z cuya duración, costo y otros requerimientos son nulos. Ahora D es sucesora inmediata de B directamente y de A por medio de Z. Asimismo, C es sucesora de A pero no de B.

### Construcción del grafo de un proyecto

Para ilustrar como se construye el grafo de un proyecto, supongamos, por ejemplo, uno con las actividades siguientes:

1. Actividad A
2. Actividad B
3. Actividad C
4. Actividad D
5. Actividad E
6. Actividad F
7. Actividad G

A continuación se tiene una tabla con los tiempos estimados para cada actividad y su listado de predecesoras inmediatas.

ACTIVIDAD #	DURACION (SEMANAS)	PREDECESORES INMEDIATOS
A	8	
B	5	
C	3	A
D	6	A
E	2	B, C
F	2	B, C
G	3	D, F

Con los datos de la tabla anterior, se procede a dibujar el grafo del proyecto representando cada actividad con un arco e introduciendo las actividades que sean necesarias para establecer las relaciones de precedencia inmediata señaladas en la tabla. El grafo se muestra en la Figura III.14. La duración de la actividad se indica entre paréntesis y la numeración de los nodos en los cuadros.

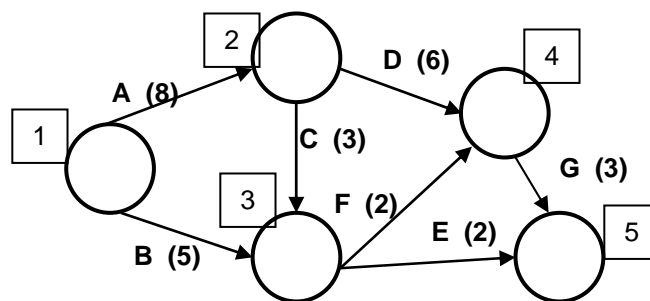


Figura III.14.- Red (grafo) de actividades



## Definición de la Ruta Crítica

A la ruta más larga entre un punto en el grafo con un arco representando una actividad que no tiene predecesores y el extremo de otro arco que representa una actividad que no tiene sucesores, se le llama la **ruta crítica**. La razón del nombre se debe a que la longitud (tiempo) de esta trayectoria es el mínimo tiempo en el cual se puede terminar el proyecto; cualquier retraso en una actividad en esta ruta alargará necesariamente todo el proyecto. Por lo anterior, es crucial asegurarse que ninguna de ellas sufra retrasos.

Las actividades que no están en la ruta crítica están holgadas en el sentido que pueden aceptar ciertos retrasos que no afectan el tiempo de ejecución del proyecto, razón por la cual no son críticas.

Para un grafo sencillo, con pocas actividades como las que se muestran en la Figura III.14 es relativamente fácil determinar “a simple vista” la ruta más larga o ruta crítica. Sin embargo, en proyectos de la vida real sus redes son más complejas, las actividades se presentan con detalle y a menudo tienen cientos o aun miles de ellas, en este caso, no es sencillo encontrar la ruta crítica.

## Determinación de la Ruta Crítica

La determinación de la ruta crítica se hace partiendo de una idea relativamente sencilla, el tiempo de inicio más próximo de una actividad es, como su nombre lo indica, el primer momento en que se puede iniciar, el cual no puede ser antes que el mayor de los tiempos de terminación más próximos de todas sus predecesoras inmediatas. Lo anterior lo podemos expresar de la siguiente manera:

$$drc(k) = \text{la máxima duración acumulada (j)} + da(k)$$

Donde  $drc(K)$  es la suma del tiempo acumulado de una ruta crítica de la actividad  $k$  que resulta de la máxima duración acumulada ( $j$ ), esta resulta de los tiempos de terminación más próximos de las actividades  $j$  que son predecesoras inmediatas de las actividades  $k$  y  $da(k)$ , la duración de la actividad  $k$ .

La forma algorítmica tradicional para resolver el problema es ir etiquetando los nodos en un arreglo, de aquellos donde se va acumulando la mayor duración resultante de la suma de las duraciones que lo preceden. Para lo anterior, es necesario, en la programación procedural, llevar apuntadores de lo realizado en la “pasada hacia delante” como se le denomina en el campo de la investigación de operaciones.

## Actividades en la Ruta Crítica

Las actividades en la ruta crítica corresponden a aquellos arcos “que las representan- cuyos extremos unen los nodos que tienen las mayores duraciones acumuladas y por medio de los cuales se unen los nodos inicial y final de la red del proyecto.

La forma algorítmica tradicional en un *software* de administración de proyectos lo hace etiquetando las actividades con un procedimiento semejante al marcaje de los nodos,

efectuando una “pasada hacia atrás” para identificar las actividades. Esto requiere, con programación procedural, llevar apuntadores para registrar lo revisado.

Otro análisis matemático es el **PERT Program Evaluation and Review Technique**, el cual usa lógica de redes secuenciales y pesos promedio estimados de duración para calcular la duración del proyecto.

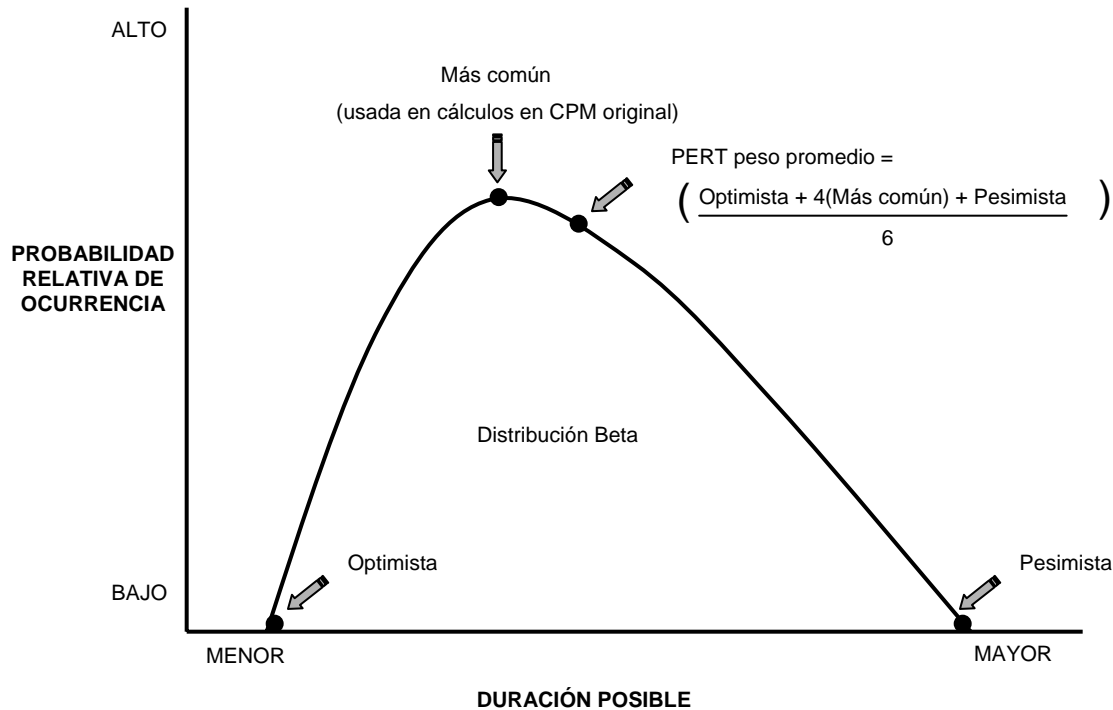


Figura III.15.- Cálculo de duración PERT

La Técnica PERT difiere de CPM principalmente en que este usa la distribución principal en lugar de la mejor de las estimaciones originalmente usada en CPM. PERT es raramente usada hoy en día aunque algunas estimaciones son usadas para cálculos en CPM. Una gráfica de cálculo PERT y CPM se muestra en la Figura III.15.

### III.3. Módulo de Conocimiento de Métricas de Desarrollo

#### III.3.1. Modelo de Productividad de *Software* de PUTNAM

En un pequeño proyecto de desarrollo de *software* una sola persona puede analizar los requerimientos, realizar el diseño, generar el código, y llevar a cabo las pruebas. A medida que el tamaño del proyecto aumenta, debe involucrarse más personas cuya cantidad es necesario estimar.

Un método que apoya en esta estimación es el **modelo de Putnam** [PUT78] que es un modelo multivariable dinámico que asume una distribución específica de esfuerzo a lo largo del ciclo de vida de un proyecto de *software*. El modelo ha sido derivado de distribuciones de mano de obra de grandes proyectos, sin embargo es posible aplicarlo a pequeños proyectos de *software*.

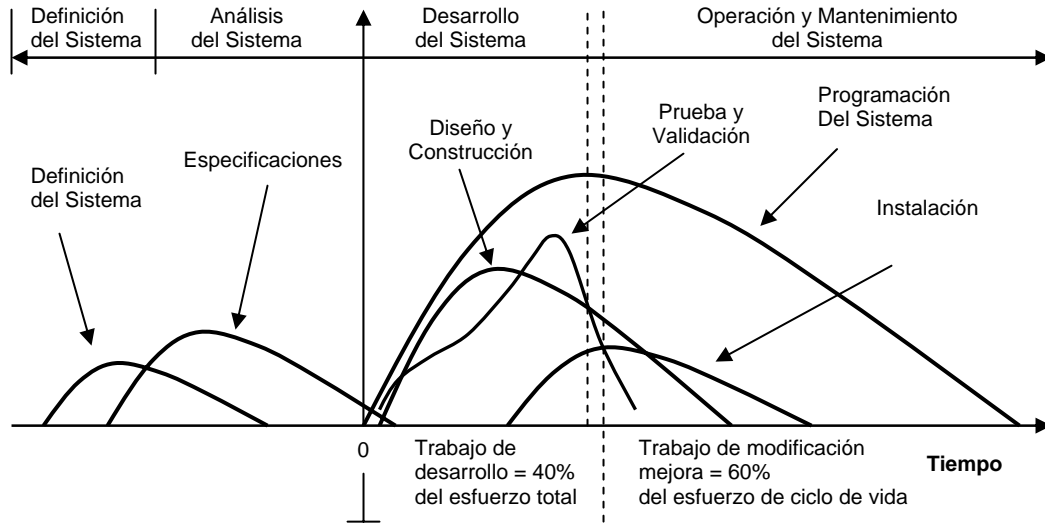


Figura III.16.- Modelo de Putnam estimado de costo y ciclo de *software*

La curva característica de este modelo se muestra en la Figura III.16, a menudo se le denomina curva de Rayleigh-Norden [PUT78] y puede utilizarse para obtener una ecuación de *software* que relacione el esfuerzo de trabajo con las líneas de código esperadas y el tiempo de desarrollo de los paquetes de trabajo:

$$K = L^3 / (Ck^3 td^4 )$$

Donde  $Ck$  es una *constante de estado de tecnología* y refleja las restricciones directas que frenan el progreso del programador. Valores típicos pueden ser:  $Ck = 2.000$  para un entorno pobre de desarrollo de *software* (por ejemplo, sin una metodología),  $Ck = 8.000$  para un buen entorno de desarrollo de *software* (con una buena metodología, modo de ejecución interactivo),  $Ck = 11.000$  para un entorno excelente (por ejemplo uso de herramientas CASE),  $K$  es el esfuerzo empleado (personas-tiempo) durante el ciclo de vida completo del desarrollo y mantenimiento del *software* en el tiempo  $td$ .

La cantidad de líneas de código  $L$  se pueden obtener utilizando una **métrica orientada al tamaño de los paquetes de trabajo**. A partir de la obtención de un número esperado como una media ponderada de las estimaciones de  $L$ : optimista(a), más probable (m), y pesimista (b).

$$L = ( a + 4m + b ) / 6$$

Esta expresión acredita a la distribución más probable y sigue una *distribución de probabilidad Beta* (Figura III.15).

## Capítulo IV Fuentes de Conocimiento y Biblioteca de Casos

### IV.1. Identificación de las Fuentes de Conocimiento

El primer paso importante en el desarrollo de un Sistema Experto es identificar a los Expertos Humanos (EH) que participaran en el proyecto. El Líder del proyecto y los Ingenieros de Conocimiento (IC) no pueden iniciar sin haber identificado las Fuentes de Conocimiento [CAI98].

Existen dos tipos de Fuentes de Conocimiento: las Activas (FCA) y las Pasivas (FCP). En la identificación de las FCP el Líder del Proyecto en forma conjunta con el grupo de Ingenieros del Conocimiento y los Expertos Humanos recopilan todo el conocimiento del dominio del Sistema Experto que se encuentra documentado en manuales, catálogos, libros de texto o cualquier otro material escrito asociado, o almacenado electrónicamente en forma de bases de datos. Posteriormente, se debe analizar el material escrito y evaluar si podrá utilizarse en el Sistema Experto.

La identificación de las Fuentes de Conocimiento Pasivas (FCP) para el desarrollo del PROTOTIPO del SE-APIS se realizó mediante la selección, clasificación y lectura de diferente bibliografía presentada en el apartado de Referencias Bibliográficas de este trabajo de tesis, obteniéndose de esta forma el conocimiento descrito en el Capítulo III sobre Ingeniería de *Software* a partir de libros de Ingeniería de *Software* (Roger S. Pressman [RPE90] y [RPE02], Ian Sommerville [IAS02]) y Administración de Proyectos que fueron definidos a partir del **Project Management Institute Standards Committee** [PMI96], e información de varios libros y paginas en Internet acerca de diversas Métricas para el desarrollo de *Software*. Adicionalmente se recopiló el conocimiento sobre el **Método de desarrollo de aplicaciones CASE (CADM)** [DES00], que se presenta en el Anexo B, siendo ésta una metodología de desarrollo moderna que incorpora herramientas CASE para el desarrollo de Sistemas.

Las Fuentes de Conocimiento Activas (FCA) se deben de seleccionar en forma conjunta con los Ingenieros de Conocimiento y varios Expertos Humanos (EH). Esta selección consiste en identificar el número de expertos y la forma en que se integrará el equipo de trabajo dependiendo de la aplicación y la magnitud del proyecto. Algunas alternativas propuestas para organizar las Fuentes de Conocimientos Activas (Expertos Humanos) son [CAI98]:

- a) Un solo EH.- Ocurre normalmente cuando la empresa cuenta con un experto único que conoce la especialidad que se va sistematizar.
- b) Una sucesión de EH.- En este caso el experto no podrá participar durante toda la vida del proyecto por lo que tendrá que ser reemplazado.
- c) Grupos de EH para resolver una tarea.- Esto ocurre cuando los diferentes expertos comparten la misma área de trabajo.
- d) EH en diferentes áreas de trabajo.- El sistema Experto que se realizará abarca diferentes áreas de trabajo, es conveniente dividir en módulos el dominio del conocimiento.
- e) Grupos de EH en diferentes áreas de trabajo.- Este esquema implica modularizar el dominio de conocimiento en donde una sociedad de expertos trabaja colaborativamente para alcanzar la solución.

En el caso particular del desarrollo del PROTOTIPO del SE-APIS, la FCA se realizó considerando la asesoría experta de los profesores del Centro de Investigación en Computación del IPN, y confiando en mi experiencia personal de varios años en la Administración y Desarrollo de proyectos de Sistemas de Información, elaboración de varias Metodologías en diferentes áreas especializadas de Informática, uso de métodos y herramientas de Ingeniería de *Software*, desarrollo de procesos integrales de negocio así como aplicación de herramientas automáticas para el desarrollo de *software*.

## IV.2. Elicitación del Conocimiento

Los Ingenieros de Conocimiento deben de considerar las diferentes técnicas para realizar la adquisición de conocimiento. Se debe estimar el número y tipo de entrevistas requeridas con los EH.

La comunicación puede realizarse individual o en forma grupal. La entrevista individual es una conversación entre dos personas dirigida directa o indirectamente, por el IC y tiene como objetivo obtener información, intercambiar ideas, observar, asesorar y/o conocer opiniones. En la entrevista grupal el IC se reúne con los integrantes del grupo de EH durante un cierto tiempo para obtener determinada información. Las técnicas que se pueden aplicar, se clasifican en: *conversacionales*, *observacionales* y *multidimensionales* [CAI98].

- a) Técnicas *conversacionales*.- son técnicas verbales apropiadas para que el experto recuerde, reflexione y explique su comportamiento, ante una situación dada, entre estas técnicas están: Entrevistas dirigidas, entrevistas no dirigidas, entrevistas estructuradas, Introspección y entrevista tutorial.
- b) Técnicas *observacionales*.- Él EH simula la solución del problema, el IC observa y registra esta solución, las técnicas más importantes son: Análisis de protocolos, la observación del experto.
- c) Técnicas *multidimensionales*.- Estas son técnicas artificiales que permiten obtener información de manera no verbal. Las técnicas a menudo, fuerzan al experto a pensar sobre el dominio de una manera diferente, las técnicas más importantes son: Selección de conceptos, ordenación de tarjetas, generación de matriz.

Considerando la situación bajo la cual se lleva a cabo un trabajo de tesis, las técnicas más apropiadas son las *multidimensionales*, ya que permiten utilizar ambos roles por parte del tesista, el de EH y el de IC, en la selección, clasificación y evaluación del conocimiento que se incluirá en el Sistema Basado en Conocimiento.

De forma particular la elicitación del conocimiento de las FCA y FCP para el SE-APIS, se llevó a cabo en un proceso de cuatro pasos, en el siguiente orden:

- a.- Lectura General.- Lectura de la Bibliografía referenciada en este trabajo de tesis considerando la terminología y conceptos formales que se incorporan en el conocimiento del Sistema Experto como Apoyo a la Administración de Proyectos de Ingeniería de *Software*.
- b.- Participación de otros EH.- Poniendo a consideración de especialistas del área de Informática de la Subdirección de Ingeniería, del grupo de Administración de Proyectos del Instituto Mexicano del Petróleo y del grupo de asesores del Centro de Investigación en Computación del IPN, el material sobre conocimientos de Ingeniería de *Software* y Administración de Proyectos.

c.- Análisis del Material.- Elaboración del análisis de la Información sobre Ingeniería de *Software* y Administración de Proyectos para poder conceptualizar un modelo que permitiera representar el conocimiento por medio de técnicas seleccionadas de la Inteligencia Artificial.

d.- Representación del Conocimiento del SE-APIS.- Selección del conocimiento para establecer la Base de Conocimientos para ser desarrollada con técnicas de Inteligencia Artificial para el Sistema Experto APIS.

### **IV.3. Biblioteca de CASOS de Metodologías de Desarrollo**

La formulación de un conjunto de casos (biblioteca) es de vital importancia para comprender las tareas que realiza el experto, para el diseño, desarrollo y la verificación y validación de los modelos. Un caso ilustra la forma en que un EH resuelve un problema. Un *conjunto de casos* ilustra una variedad de situaciones diferentes que el experto puede manejar. Una *biblioteca de casos* [CAI98] incluye *casos prototipo* del dominio y *casos excepcionales* del mismo. Los primeros representan aquellos casos que ocurren con frecuencia y que su solución es aplicable a nuevas situaciones agregando adaptaciones. Las excepciones representan aquellos casos conflictivos, que comparten ciertas características con los prototipos, y que sin embargo requieren un tratamiento diferente por poseer algún atributo especial. Los casos deben ser proporcionados por los expertos humanos (FCA), aunque también se pueden obtener de los registros que conservan las organizaciones (FCP).

#### **Casos generales de Metodologías de Software:**

De acuerdo con los diferentes modelos y ciclos mostrados en el Capítulo III, existen varios casos de metodologías, sin embargo el *caso prototipo* es que al menos ésta debe contar con las fases básicas de *análisis, diseño, construcción y pruebas* considerando los recursos que serán utilizados así como el medio ambiente cambiante bajo el cual operará el sistema desarrollado y que obviamente requerirá que se realicen correcciones y cambios en los programas.

Para apoyar lo anterior el caso prototipo debe incorporar la fase de *planeación*, la cual se vuelve muy importante en el desarrollo de sistemas de información complejos para estimar los recursos humanos, materiales y financieros que la organización requerirá para llevar a cabo los proyectos de ingeniería de *software* con el tiempo, costo y calidad requeridos de acuerdo con la plataforma de tecnología utilizada. Adicionalmente las metodologías actuales consideran dentro del ciclo de desarrollo de sistemas la puesta en marcha y la operación e incorporan la fase de *mantenimiento del sistema*, como por ejemplo en la metodología del *modelo b*.

Es así, que las fases que están perfectamente delimitadas en un caso prototipo de una metodología dentro de un paradigma aceptado actualmente son: *planeación, análisis, diseño, construcción, pruebas y mantenimiento*, las cuales pueden ser usadas en diferentes modelos que se desarrollan en forma secuencial e iterativamente.

Los sistemas más complejos son los que requieren casos de modelos de metodologías más completos como el *modelo de espiral* que incluye todas las fases mencionadas y adicionalmente dentro de los diferentes ciclos de la espiral se manejan *modelos de prototipos de software* que permiten obtener con una mayor facilidad los requerimientos del usuario.

### Caso específico de la Metodología CADM:

Considerando los conceptos anteriores un caso específico seleccionado de una metodología para ser incorporado en la base de conocimientos del SE-HIS es el **Método de desarrollo de aplicaciones con CASE** (CADM, [DES00]) mostrado en la Figura IV.1, el cual reúne formalmente todas las fases de una metodología actual para desarrollo de sistemas de información de cualquier tamaño y complejidad, además de que enfoca sus conceptos de manera directa en el uso de herramientas CASE (Anexo B).

Método Prototipo	De desarrollo de aplicaciones CASE
FASE	ACTIVIDAD
Estrategia (Planeación)	Objetivo y organización del negocio. Modelo E-R de estrategia Alcance del sistema y análisis de costo beneficio Documento de Estrategia. Plan de trabajo ejecutivo. Modelo conceptual de procesos del negocio. Evaluación de estrategia.
Preanálisis	Plan de análisis. Plan para implementación de los estándares CASE de análisis. Evaluación del preanálisis.
Análisis	Diagrama entidad relación de análisis. Flujo de procesos lógicos. Documento de requisitos. Evaluación del análisis. Reingeniería de procesos del negocio. Evaluación del análisis. Documentación de análisis.
Prediseño	Plan de diseño. Flujo de procesos físicos. Estándares de diseño. Prototipo conceptual de pantallas. Evaluación de prediseño.
Diseño	Diseño de la base de datos. Diagrama E-R de diseño. Diseño de aplicaciones. Diseño de pantallas (prototipo). Libro de diseño. Evaluación del diseño. Documentación de diseño
Construcción.	Construcción de la base de datos. Programación de las aplicaciones. Pruebas unitarias Documentación de código.
Pruebas y control de cambios	Control de cambios durante cada fase del proceso. Plan de pruebas. Aceptación del usuario. Documentación de pruebas.
Implementación	Plan de implementación. Revisión de documentación. Capacitación de usuarios. Puesta en marcha.
Mantenimiento	Corrección de errores. Modificación de objetos. Creación de nuevos elementos. Documentación de cambios.

Figura IV.1.- Caso específico prototipo CADM

## Capítulo V. Técnicas de la Inteligencia Artificial para el desarrollo del SE-APIS

El desarrollo de un Sistema Experto requiere de la aplicación de diversas técnicas de la Inteligencia Artificial. En particular el Sistema Basado en Conocimiento como Apoyo para la Administración de Proyectos de Ingeniería de *Software* HIS, SE-APIS, debido a su alcance aplicará cuatro técnicas: Representación del Conocimiento, Estructuras de ranura y relleno débiles o *Marcos*, Estructuras de ranura de relleno fuertes o Dependencias Conceptuales (CD) y Procesamiento de Lenguaje Natural, cuyos conceptos fundamentales y estructuras asociadas se presentan en los apartados de este capítulo.

### V.1. Representación del conocimiento

Para resolver los complejos problemas con los que se enfrenta la inteligencia artificial, es necesario disponer tanto de una gran cantidad de conocimiento como de una serie de mecanismos que permitan manipularlo con el fin de obtener soluciones a nuevos problemas. En los problemas de **IA** se representa el conocimiento de muy distintas maneras.

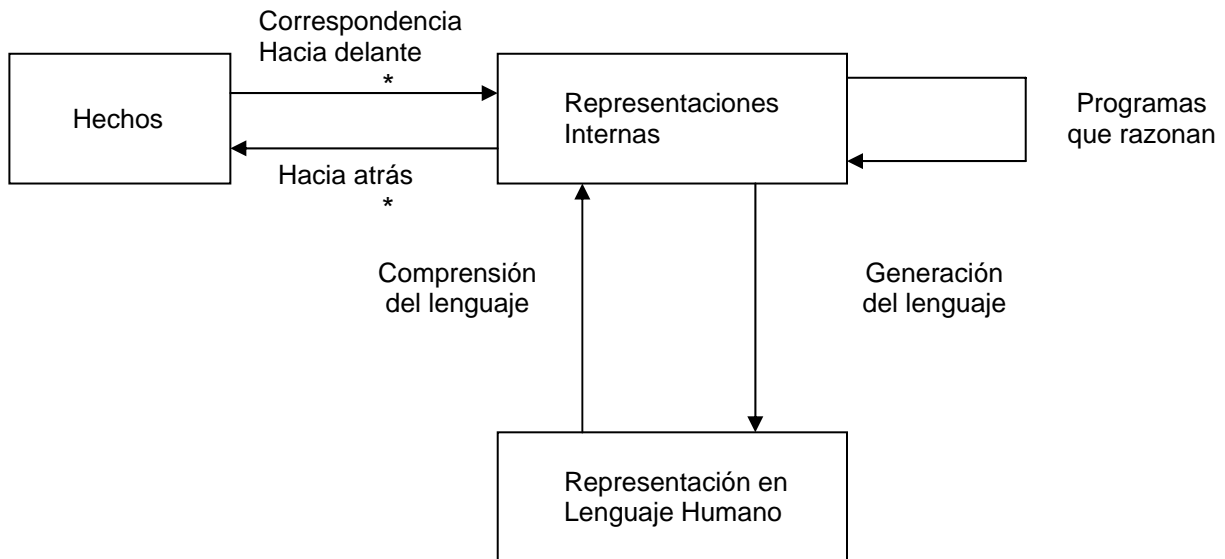


Figura V.1.- Representación del conocimiento

De acuerdo con Newell [NEW82] es posible estructurar el conocimiento en dos niveles distintos:

- El nivel del conocimiento, donde se describen los hechos (incluyendo el comportamiento y los objetivos del Sistema Experto).
- El nivel simbólico, donde se describen los objetos de nivel del conocimiento en términos de símbolos manipulables por programas.

Estos dos niveles operan de acuerdo a la Figura V.1. La representación hacia adelante establece una correspondencia entre los hechos y sus representaciones, mientras que la representación hacia atrás establece la correspondencia inversa, desde las representaciones a los hechos.



Existe una representación de los hechos que, por lo extendido de su uso, merece una mención especial; las frases en **Lenguaje Natural (LN)**. Independientemente de la forma en que representemos los hechos en un programa, es posible que sea necesario tener en cuenta la representación de esos hechos en castellano para facilitar así el intercambio de información con el programa. En este caso, serán necesarias unas funciones de correspondencia que transformen las frases en castellano en la representación que se vaya a utilizar y viceversa.

Es importante recordar que las funciones de correspondencia no suelen ser biunívocas. De hecho, no suelen ser ni siquiera funciones, sino relaciones de muchos (en otras palabras, a cada elemento del dominio le suelen corresponder varios elementos del rango, y diferentes elementos del dominio se pueden corresponder con un mismo elemento del rango). Esto se hace especialmente patente cuando en la correspondencia están involucradas representaciones de hechos en lenguaje natural. Por ejemplo, las frases: “Todos los perros tienen rabo” y “Todo perro tiene un rabo” pueden representar el mismo hecho, que todos los perros tienen al menos un rabo

### **V.1.1. Aproximaciones a la representación del conocimiento**

Un buen sistema de representación del conocimiento en un dominio particular debe poseer las siguientes propiedades:

- Suficiencia de la representación: La capacidad de representar todos los tipos de conocimiento necesarios en el dominio.
- Suficiencia deductiva: La capacidad para manipular las estructuras de la representación con el fin de obtener nuevas estructuras que se correspondan con un nuevo conocimiento deducido a partir del antiguo.
- Eficiencia deductiva: La capacidad de incorporar información adicional en las estructuras de conocimiento con el fin de que los mecanismos de inferencia puedan seguir las direcciones más prometedoras.
- Eficiencia en la adquisición: La capacidad de adquirir nueva información con facilidad. El caso más simple es aquél en el que una persona inserta directamente el conocimiento en la base de datos. Idealmente, el programa sería capaz de controlar la adquisición de conocimiento por sí mismo.

Desgraciadamente todavía no se ha encontrado ningún sistema que optimice todos estos aspectos y que sea aplicable a cualquier tipo de conocimiento. En consecuencia, existen múltiples técnicas para la representación del conocimiento. Muchos programas utilizan más de una técnica.

### **Conocimiento relacional simple**

El modo más sencillo de representar los hechos declarativos es mediante un conjunto de relaciones del mismo tipo que las utilizadas en los sistemas de bases de datos. En la Figura V.2 se muestra un ejemplo de dichos sistemas relacionales. Se dice que esta representación es simple debido a la escasa capacidad deductiva que ofrece. Pero el conocimiento representado de esta forma puede servir como entrada a otros mecanismos de inferencia más potentes.

Los sistemas de bases de datos están diseñados para proporcionar el soporte adecuado a este tipo de conocimiento.

Etapa	Duración	Peso	Complejidad
Análisis	6	.20	Baja
Diseño	6	.30	Media
Construcción	5	.30	Alta
Pruebas	2	.20	Alta

Figura V.2.- Conocimiento relacional simple

### Conocimiento heredable

El conocimiento relacional de la Figura V.2 se compone de un conjunto de atributos que junto con unos valores asociados permite describir los objetos de la base de conocimiento. Es posible extender la representación básica con los mecanismos de inferencia que se desee. Una de las formas más útiles de inferencia es la **herencia de propiedades**, donde los elementos de una clase heredan los atributos y los valores de otras clases más generales en las que están incluidos.

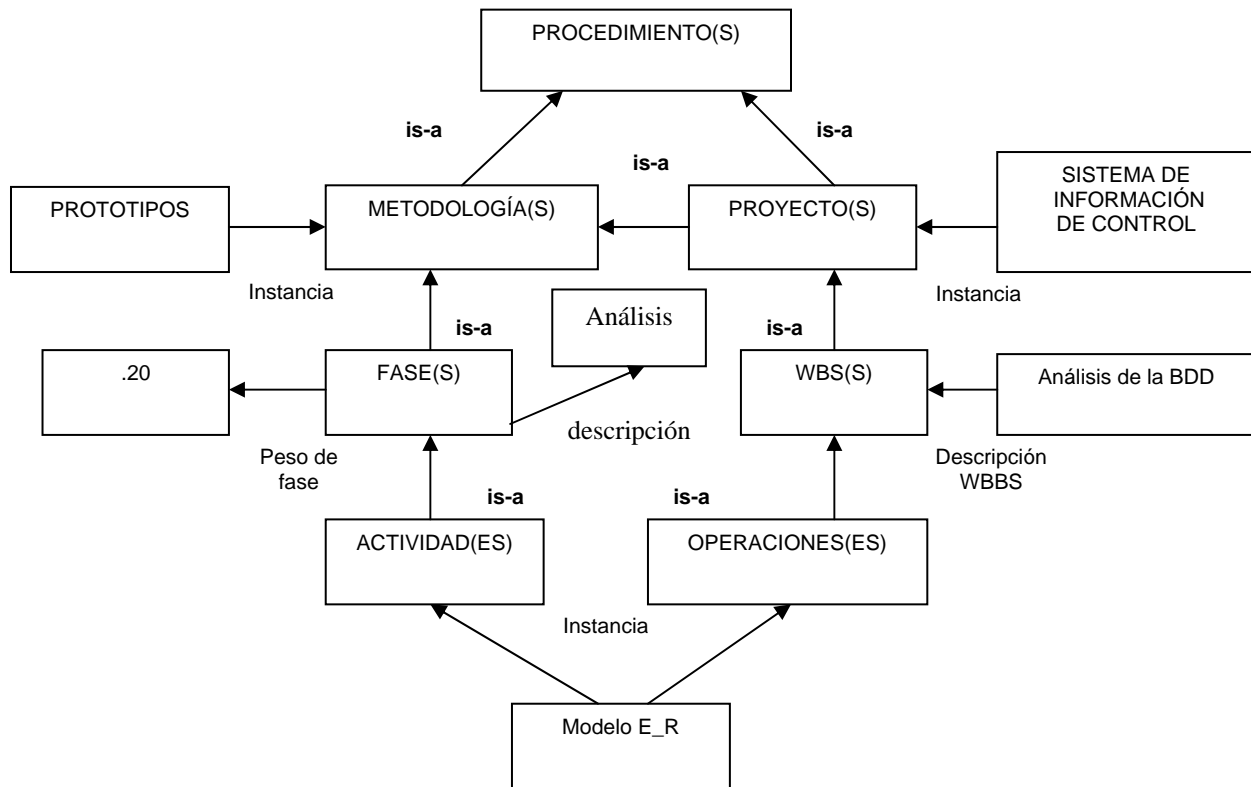


Figura V.3.- Conocimiento heredable

Para dar soporte a la herencia de propiedades, los objetos se deben organizar en clases, y las clases se deben disponer como una jerarquía de generalizaciones. En la Figura V.3 aparece una estructura organizada de esta forma, en ella se ha introducido conocimiento acerca de Ingeniería de *Software*. Las líneas representan atributos de los objetos. Estos valores, a su vez, también se pueden ver como objetos con atributos y valores, y así sucesivamente. Las flechas conectan los objetos con sus valores a través de los correspondientes atributos. La estructura que aparece en la figura es lo que se denomina una **estructura de ranura y relleno** (slot-and-filler). También se puede denominar **red semántica** o colección de estructuras **MARCOS**.

La Figura V.4 muestra el nodo correspondiente a una fase de metodología representada como una estructura.

FASES	
Es un	Metodología
Peso de Fase	.20
Descripción	Análisis

Figura V.4.- Un nodo visto como un marco (frame)

Los objetos y los atributos que se utilizan en este ejemplo pertenecen al dominio de la Ingeniería de *Software*. Las dos excepciones son el **atributo es un (is-a)**, que se utiliza para indicar que una clase está contenida en otra, y el **atributo instancia**, que se utiliza para indicar pertenencia a una clase. Estos dos atributos específicos son la base de la herencia de propiedades como una técnica de inferencia.

### Conocimiento deductivo

La herencia de propiedades es una forma muy potente de inferencia, pero no es la única. En algunas ocasiones es necesario echar mano de toda la potencia de la programación lógica, PROLOG (Clockin y Milleth, 1984 [CMI84]) es el más popular para describir las inferencias apropiadas. En la Figura V.5 se muestra un ejemplo de la lógica de predicados de primer orden aplicada a la representación de conocimiento adicional acerca de la de Ingeniería de *Software*.

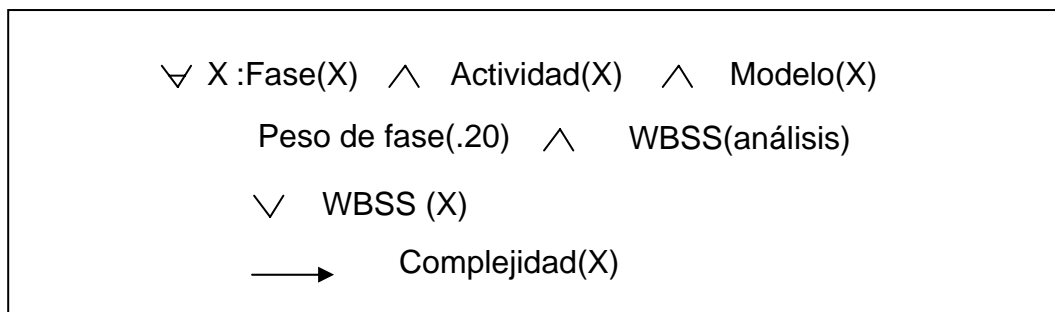


Figura V.5.- Conocimiento deductivo.

Por supuesto, este conocimiento será inútil amenos que se disponga de un mecanismo de inferencia que lo pueda aprovechar. El procedimiento de inferencia necesario en este caso será uno que implemente las reglas lógicas de la inferencia. Existen muchos de tales mecanismos, algunos de los cuales razonan hacia delante a partir de los hechos hasta llegar a las conclusiones, mientras que otros razonan hacia atrás desde las conclusiones buscadas hasta los hechos de partida.

### Conocimiento procedimental

El conocimiento procedimental se puede representar en los programas de muchas maneras distintas. Por ejemplo las construcciones de programación en Dijkstra [1976], Hoare[1985], y Chandy y Misra [1989]. La manera más habitual consiste en especificarlo como un código que hace algo. La máquina utiliza el conocimiento cuando ejecuta el código para llevar a cabo una determinada tarea. Desgraciadamente, esta forma de representación del conocimiento procedimental es poco adecuada en términos de suficiencia deductiva.

La técnica de representación del conocimiento procedimental más utilizada en los programas de IA consiste en el uso de reglas de producción: La Figura V.6 muestra un ejemplo de regla de producción que representa parte del conocimiento operacional que suele poseer la ingeniería de *software*.

Sí:	Procedimiento metodología, y Metodología CADM, y Fase de análisis, y Peso de fase .20, y Actividad análisis de base de datos
Entonces:	Modelo es Base de Datos E_R.

Figura V.6.- Conocimiento procedimental expresado en forma de reglas

### V.1.2. Problemas de la representación del conocimiento

Los problemas para representar el conocimiento en los mecanismos deben considerar los siguientes puntos:

- Si existen atributos tan genéricos que aparecen en prácticamente todos los dominios de aplicación, es necesario asegurarse que sean tratados adecuadamente en cada uno de los mecanismos que se propongan.
- Deben existir relaciones relevantes entre los atributos de los objetos.
- Definir el nivel de representación del conocimiento e identificar el conjunto de primitivas que permita una descomposición adecuada del conocimiento.
- Analizar la estructura de representación de los conjuntos de objetos.
- Dada una base de conocimiento muy extensa, estructurar la forma de acceder a los fragmentos relevantes en cada conocimiento.

## Relaciones entre atributos

Los atributos que se utilizan para describir a los objetos pueden ser a su vez entidades representables y poseen las siguientes propiedades

- Inversos
- Existencia en una jerarquía is-a
- Técnicas para el razonamiento acerca de los valores
- Atributos univaluados.

### Inversos

Las entidades del mundo se pueden relacionar de muy diversas maneras, pero desde el momento en que decidimos describir esas relaciones como atributos, nos restringimos a una perspectiva según la cual nos fijamos en un objeto y tratamos de establecer las relaciones que existen entre él y los otros.

Así por ejemplo, en la Figura V.3 se utilizaban los atributos **instancia** e **is-a**. Cada uno de ellos aparece en la figura junto a una flecha que parte del objeto a describir y llega al objeto que representa el valor del atributo en cuestión. Pero de la misma forma nos podríamos haber centrado en el objeto que representa al valor. Si se hace así, aun se podría establecer una relación entre las dos entidades, aunque sería una relación diferente.

### Una jerarquía de atributos is-a

De la misma forma que existen clases de objetos y subconjuntos más específicos de esas clases, también se puede hablar de atributos y especializaciones de los atributos. Considérese por ejemplo, el atributo peso de fase. Este atributo es en realidad una especialización del atributo complejidad, que a su vez, es una especialización de atributo-fase. Este tipo de relaciones de generalización-especialización referidas a los atributos tienen la misma misión que cuando se aplican a los demás conceptos, sirven de soporte a la herencia. En el caso de los atributos, la información que se hereda consiste en cosas tales como restricciones sobre los valores que un atributo puede tomar o mecanismos para el cómputo de dichos valores.

### Técnicas para el razonamiento acerca de los valores

En ocasiones los valores de los atributos se especifican explícitamente durante la creación de una base de conocimiento. En el ejemplo sobre Ingeniería de *software* de la Figura V.3 aparecen varias definiciones de este tipo. Pero generalmente el sistema debe razonar sobre valores que no ha recibido explícitamente. Hay varios tipos de información que pueden jugar un determinado papel en este razonamiento, incluyendo:

- Información acerca del tipo de valor.
- Restricciones sobre el valor, que suelen venir expresadas en términos de entidades relacionadas.
- Reglas para el cómputo de un valor: reglas hacia atrás o también reglas si-necesario.
- Reglas que describen las acciones que se deberían llevar a cabo en el caso de que se llegase a conocer un determinado valor. Estas reglas se denominan hacia adelante, o en ocasiones reglas sí-añadido.

## Atributos univaluados

Un tipo de atributo, no por específico menos útil, es aquel que sólo puede tomar un único valor. En los sistemas de representación del conocimiento se han seguido diferentes aproximaciones en el tratamiento de los atributos univaluados, incluyendo:

- La introducción de una notación explícita para los intervalos, temporales. Si dos valores diferentes coinciden en el mismo intervalo de tiempo se avisa automáticamente de que se ha producido una contradicción.
- Suponer que el único intervalo de tiempo relevante es el momento actual. De manera que si se asigna un nuevo valor, éste sustituye al antiguo.
- Ningún mecanismo específico. Los sistemas basados en la lógica son de este tipo. Pero en estos sistemas es posible definir axiomas que identifican a un atributo como univaluado de manera que es sabido que no puede tomar otros valores.

### V.1.3. Selección de la granularidad de la representación

Independientemente del mecanismo de representación que se elija, es necesario definir a qué nivel de detalle se debería representar el conocimiento y también formular las primitivas que se deben utilizar. Estableciendo si se debe hacer un reducido número de primitivas de bajo nivel o se debe hacer un conjunto más grande que cubra un rango más amplio de granularidades. Un breve ejemplo servirá de ilustración. Supongamos que estamos interesados en representar el hecho:

Tutor consulta una metodología. Una posible representación sería:

Consulta(agente(Tutor),objeto(metodología)).

Distintos programas de **IA**, entre los que se incluyen los trabajos de Schank y Abelson [SHA77] y los de Wilks [WIL72], están basados en el uso de bases de conocimiento descritas en términos de un pequeño conjunto de primitivas. El convertir todas las afirmaciones a una representación que sólo utiliza un pequeño conjunto de primitivas tiene la ventaja fundamental de que se pueden escribir todas las reglas de inferencia en términos de esas primitivas, en lugar de tener que considerar todas las formas posibles en que podría estar expresado el conocimiento originalmente.

## V.2. Estructuras de ranura y relleno

### V.2.1. Estructuras de ranura y relleno débiles: **MARCOS**

Estas estructuras se introdujeron como un dispositivo para soportar adecuadamente la herencia a lo largo de los enlaces **is-a** e **instancia**. Éste es un importante aspecto de estas estructuras. La herencia monótona (Nilsson, 1980 [NIL80]) se puede desarrollar más eficazmente con estas estructuras que con la lógica pura, y la herencia no monótona [Quine y Ullian, 1978 [QUL78]), puede soportarse muy fácilmente. La razón por la que la herencia se ejecuta de un modo sencillo, es que en los sistemas de ranura y relleno el conocimiento está estructurado como un conjunto de entidades y todos sus atributos. Esta estructura tiene una gran utilidad además de ser el soporte de la herencia, por las siguientes razones:

- Indiza las aserciones en orden de las entidades que describen. De un modo más formal, indiza los predicados binarios, en función de su primer argumento. Como resultado de esto, se puede recuperar el valor de un atributo de una determinada entidad de un modo mucho más rápido.
- Hace que la descripción de las propiedades de las relaciones sea mucho más sencilla. Para hacer esto en un sistema lógico puro se requieren mecanismos de ordenación altamente estructurados.
- Es una forma de programación orientada a objetos, con todas las ventajas que ello acarrea, incluyendo la modularidad, así como la relativa facilidad para verlos desde un primer momento.

Existen dos enfoques de este tipo de estructuras: las **redes semánticas** (semantic nets) y los **marcos** (frames).

#### **MARCOS**

Un **marco** (frame) es una colección de atributos, normalmente llamados ranuras (**slots**), con valores asociados (y posibles restricciones entre los valores), que describe alguna entidad del mundo. Algunas veces el marco describe una entidad en un sentido absoluto, y otras representa la entidad desde un punto de vista particular (como por ejemplo en la propuesta de sistema de visión (Minsky, 1975 [MIN75]), en el cual se introdujo por primera vez el término marco). Un marco único, tomado independientemente, no suele ser útil. En lugar de eso se construyen sistemas de **marcos** a partir de colecciones de **marcos** conectados unos con otros en virtud del hecho de que el valor de un atributo de un marco puede ser a su vez otro marco.

#### **Los marcos como conjunto e instancias**

La teoría de conjuntos proporciona una buena base para comprender los sistemas de **marcos**. Aunque no todos los sistemas de **marcos** se definen de este modo. Con este enfoque, cada marco representa, ya una clase (un conjunto), ya una instancia (un elemento de la clase). Para ver cómo funciona esto, se considerará el sistema de **marcos** que se muestra en la Figura V.7, que es una forma ligeramente modificada de la red que se mostró en la Figura.V.3. En este ejemplo los **marcos** Procedimiento, Metodología, Fases, Proyectos son todas las clases. Los **marcos** WBSS y Sistemas de Información de Control son instancias. La relación **is-a**, es la relación subconjunto.

Debido a que una clase representa un conjunto, existen dos clases de atributos que se pueden asociar con ésta: atributos acerca del conjunto en sí mismo, y atributos para ser heredados por cada elemento del conjunto. Se indica la diferencia entre estos dos tipos asociando al segundo un asterisco (\*).

Existen dos tipos de clases: las regulares cuyos elementos son entidades individuales y las metaclasses que son unas clases especiales cuyos elementos son clases en sí mismos. Así una clase es ahora un elemento (instancia) de alguna clase, así como un subconjunto (**is-a**) de una o más clases. Una clase hereda propiedades de la clase de la cual es una instancia, exactamente del mismo modo que lo hace cualquier instancia. Además, una clase traslada propiedades hereditarias desde sus superclases hacia sus instancias.

<b>Metodología:</b>	
Is-a	Procedimiento
Cardinalidad:	5
* descripción:	Prototipos
<b>Fases:</b>	
Is-a	Metodología
Cardinalidad:	5-9
* Peso:	.20
* fase::	estrategia del negocio.
<b>Proyectos:</b>	
Is-a:	Procedimiento
Cardinalidad:	100000
* identificación:	P.00001
<b>WBSS:</b>	
Instancia :	Fases
Fecha inicio	04/01/99
Trabajo:	3000
Duración:	309
Proyecto:	P.00002
<b>Sistemas de Información de Control:</b>	
Instancia :	Proyectos
Tamaño en LC :	8000
Responsable:	Felipe Juárez
Duración:	600

Figura V.7.- Un sistema de *marcos* simplificado

En todos los sistemas de *marcos* que se han visto, todas las clases son instancias de la metaclassa Clase. Por tanto, todas tienen el atributo cardinalidad. En general no se utilizará la clase Clase ni el atributo cardinalidad, para las descripciones de las instancias, a menos que exista una razón concreta para incluirlos.

Toda clase es un conjunto, pero no todos los conjuntos serán descritos necesariamente como una clase. Una clase describe un conjunto de entidades que comparten propiedades significativas. En particular la información por defecto asociada con una clase se puede utilizar como una base para valores inferencia para las propiedades de sus elementos individuales.



## V.2.2. Estructuras de ranura y relleno fuertes: Dependencias Conceptuales

Las estructuras de ranura y relleno descritas en el punto anterior son muy generales. Las redes semánticas y los sistemas de *marcos* individuales pueden tener enlaces y procedimientos de inferencia especializados, pero no existen reglas complejas y rápidas sobre el tipo de objetos y enlaces que son en general adecuados para representar el conocimiento. Estas decisiones se dejan para el constructor del sistema de *marcos*.

La estructura que se explica en este punto, dependencia conceptual **CD** (conceptual dependency), incorpora nociones específicas sobre los tipos de objetos y relaciones que están permitidos. Representan poderosas teorías sobre la forma en que los programas de IA pueden representar y utilizar el conocimiento sobre situaciones comunes.

### Dependencia conceptual - CD

La dependencia conceptual (con frecuencia abreviada en inglés como **CD**), es una teoría sobre la representación del tipo de conocimientos sobre los eventos que normalmente aparecen en las frases de lenguaje natural. El objetivo consiste en representar el conocimiento de alguna forma que:

- Facilite extraer inferencias de las frases.
- Sea independiente del lenguaje en el que estén las frases originalmente.

Debido a estos dos requisitos, la representación en **CD** de una frase no se construye con primitivas que se corresponden con las palabras que aparecen en la frase, sino con primitivas conceptuales que pueden combinarse para formar el significado de las palabras en cualquier lenguaje concreto. Esta teoría la describió Schank en 1973 [SHA73] y más tarde fue desarrollada en Schank en 1975 [SHA75]. Se ha implementado en varios programas que leen y comprenden texto en lenguaje natural. Al contrario que en las redes semánticas, que proporcionan sólo una estructura en la que pueden situarse nodos que representan información a cualquier nivel, la dependencia conceptual proporciona tanto una estructura como un conjunto específico de primitivas, a un nivel concreto de granularidad, en las que pueden construirse representaciones de trozos particulares de información.

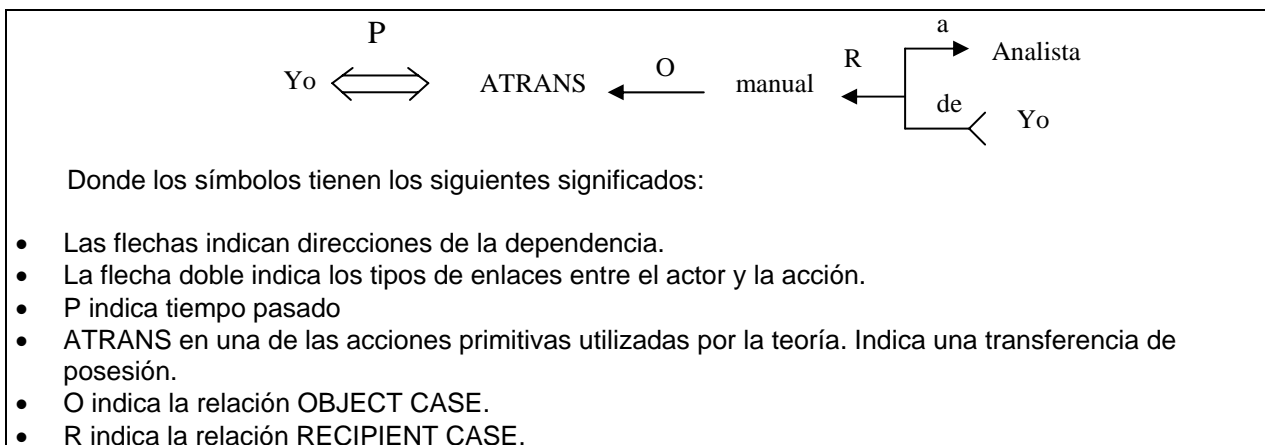


Figura V.8.- Una sencilla representación en Dependencia Conceptual.

La Figura V.8 muestra un ejemplo sencillo de la forma en que se representa el conocimiento en CD para la frase.

(Le di un manual al analista)

En **CD**, las representaciones de las acciones se construyen a partir de un conjunto de acciones primitivas. Aunque existen diferencias significativas sobre el conjunto exacto de acciones primitivas que proporcionan las distintas implementaciones de CD, un típico conjunto, tomado de Schank y Abelson (1977 [SHA77]) es el siguiente:

ATRANS	Transferencias de una relación abstracta (p.ej., dar)
PTRANS	Transferencia de una localización física de un objeto (p.ej., ir)
PROPEL	Aplicación de fuerza física a un objeto (p.ej., empujar)
MOVE	Movimiento de una parte del cuerpo por su dueño (p.ej., patear)
GRASP	Asimiento de un objeto por un acto (p.ej., empuñar)
INGEST	Ingestión de un objeto por parte de un animal (p.ej., comer)
EXPEL	Expulsión de algo del cuerpo de un animal (p.ej., llorar)
MTRANS	Transferencia de información mental (p.ej., decir)
MBUILD	Construcción de información nueva aparte de la existente (p.ej., decidir)
SPEAK	Producción de sonidos (p.ej., hablar)
ATTEND	Concentración de un órgano sensorial hacia un estímulo (p.ej., escuchar).

Un segundo conjunto de bloques construidos de **CD** es el conjunto de las dependencias permitidas entre las conceptualizaciones descritas en una frase. Existen cuatro categorías conceptuales primitivas a partir de las cuales pueden construirse estructuras de dependencias. Estas son:

ACTs	Acciones
PPs	Objetos (productores de imágenes)
AAs	Modificaciones de acciones (asistentes de acciones)
PAs	Modificaciones de PPs (asistentes de imágenes)

Además, las estructuras de dependencia son en sí mismas conceptualizaciones y pueden servir como componentes de estructura de dependencias más grandes.

Las dependencias entre conceptualizaciones se corresponden con las relaciones semánticas entre los conceptos subyacentes. En las Figuras V.9(a y b) se proporciona una lista de las más importantes que permite la CD.

La primera columna contiene las reglas, la segunda contiene ejemplos de su uso, y la tercera contiene una versión en inglés de cada ejemplo. Las reglas mostradas en la figura pueden interpretarse de la siguiente manera:

- La regla 1 describe la relación entre un actor y el evento que él o ella causa. Ésta es una dependencia a “dos vías” ya que ni el actor ni el evento pueden considerarse primarios. La letra p sobre el enlace de la dependencia indica tiempo pasado.
- La regla 2 describe la relación entre un PP y PA que se asertan para describirla. Muchas descripciones de estado, como la altura, se representa en CD mediante una escala numérica.
- La regla 3 describe la relación entre dos PP, uno de los cuales pertenece al conjunto definido por el otro.
- La regla 4 describe la relación entre un PP y un atributo que ya ha sido predicado de él. La dirección de la flecha va hacia el PP descrito.

- La regla 5 describe la relación entre dos PP, uno de los cuales proporciona algún tipo de información sobre el otro. Los tres tipos de información más usuales son la posesión (mostrado como POSEIDO-POR), la situación (mostrado como LOC) y la contención física (mostrado como CONT). La dirección de la flecha de nuevo es hacia el concepto que se describe.
- La regla 6 describe la relación entre un ACT y el PP objeto del ACT. La dirección de la flecha es hacia el ACT, ya que el contexto del ACT específico determina el significado de la relación del objeto.
- La regla 7 describe la relación entre un ACT y la fuente y el recipiente del ACT.
- La regla 8 describe la relación entre un ACT y el instrumento con el que se lleva a cabo. El instrumento siempre debe ser una conceptualización completa (es decir, debe contener un ACT), no sólo un único objeto físico.

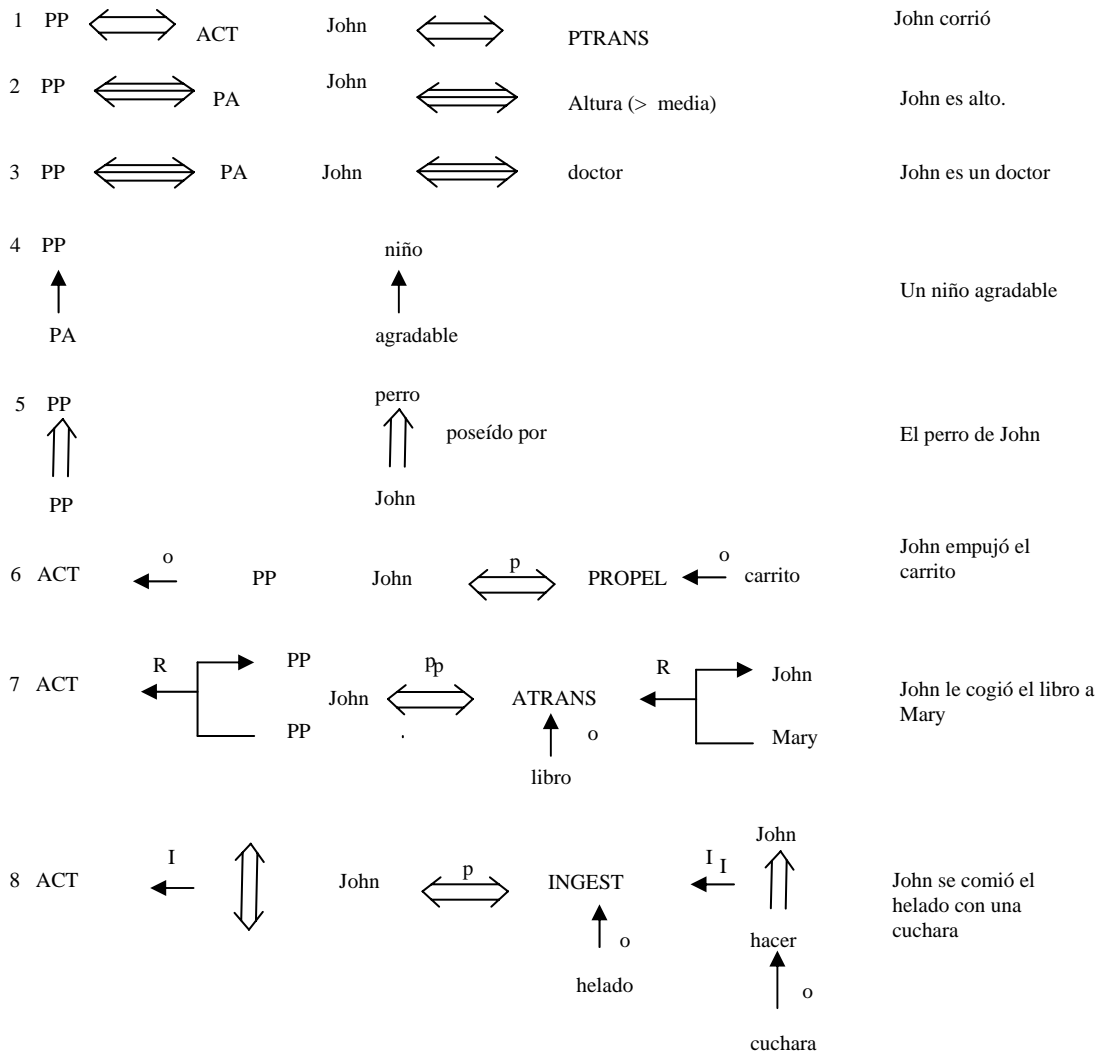


Figura V.9a.- Las dependencias de la Dependencia conceptual (CD)

- La regla 9 describe la relación entre un ACT y su fuente y destino físico.
- La regla 10 representa la relación entre un PP y el estado de comienzo y otro de destino.

- La regla 11 describe la relación entre una conceptualización y el tiempo en el que ocurre el evento descrito.
- La regla 12 describe la relación entre una conceptualización y otra que se produce a la vez. El ejemplo de esta regla también muestra cómo CD utiliza un modelo del sistema humano de procesamiento de la información; VER se representa como la transferencia de información entre los ojos y el procesador de consciencia.
- La regla 13 describe la relación entre una conceptualización y el sitio en el que ocurre.

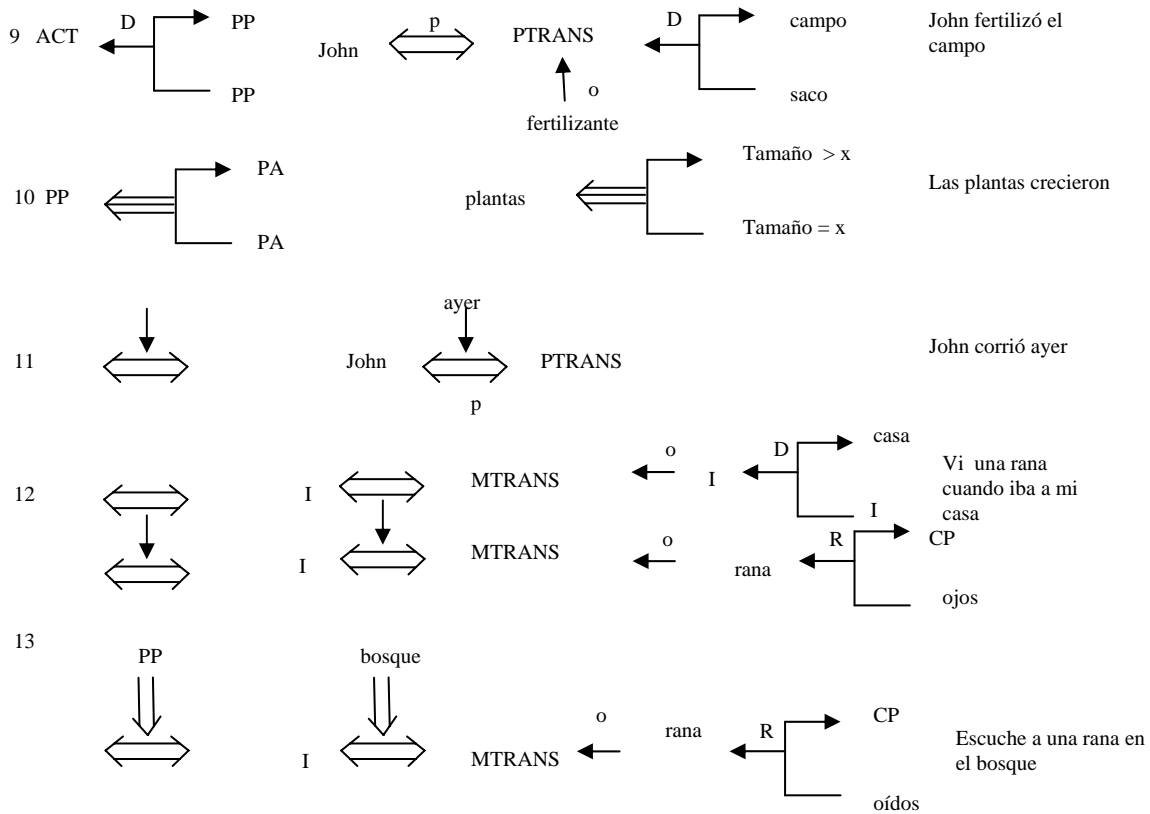


Figura V.9b.- Las dependencias de la Dependencia conceptual (CD)

Existen tres importantes formas en las que la representación del conocimiento mediante el modelo de dependencias conceptuales facilita el razonamiento con ese conocimiento:

1. Se necesitan menos reglas de inferencia de las que serían necesarias si el conocimiento no estuviera clasificado en primitivas.
2. Muchas inferencias están ya contenidas en la representación misma.
3. La estructura inicial que se construye para representar la información contenida en una frase tendrá huecos que será necesario rellenar. Estos huecos pueden servir como centros de atención para el programa que debe comprender las frases resultantes.

El primer argumento a favor de representar el conocimiento en términos de primitivas de CD en lugar de hacerlo en términos de más alto nivel en los que se describe normalmente, es que el uso de primitivas hace que sea más sencillo describir las reglas de inferencia mediante las cuales el conocimiento puede manipularse.

Un segundo argumento a favor del uso de la representación **CD** consiste en que para construirla no sólo debe utilizarse la información que explícitamente se establece en una frase, sino que también debe usarse un conjunto de reglas de inferencia asociadas con la información específica. Una vez aplicadas estas reglas, se almacenan estos resultados como parte de la representación y, por lo tanto, pueden usarse repetidas veces sin que vuelvan a aplicarse las reglas.

### **V.3. Procesamiento del Lenguaje Natural**

El lenguaje es un medio para comunicarse con el mundo. Estudiando el lenguaje seremos capaces de comprender más cosas sobre él: podemos comprobar nuestras teorías acerca del mundo viendo la forma en que apoyan nuestro intento de comprender el lenguaje. Si se tiene éxito al construir un modelo computacional del lenguaje, tendremos a nuestra disposición una potente herramienta de comunicación. En este capítulo, describimos la forma de utilizar el conocimiento en combinación con hechos lingüísticos para construir sistemas las reglas computacionales de lenguaje natural basados en Allen (1987 [ALL87]), Cullingford (1986 [CUL86]), Dowty (1985 [DOW85]) y Grosz (1986 [GRO86]).

#### **Proceso de Lenguaje Natural**

Los pasos que componen el proceso de comprensión de lenguaje natural (McKeown y Swartout (1987 [MCS87]), y McDonald y Bolc (1988 [MCB88]) son los siguientes:

- **Análisis morfológico:** Se analizan los componentes de las palabras individuales, y se separan de las palabras los constituyentes que no forman parte de ellas, como los símbolos de puntuación.
- **Análisis sintáctico:** Se transforman las secuencias lineales de palabras en ciertas estructuras que muestran la forma en que las palabras se relacionan entre sí. Se pueden rechazar algunas secuencias de palabras si infringen las reglas del lenguaje sobre la forma en que las palabras pueden combinarse. Por ejemplo, un analizador sintáctico de castellano rechazaría la frase “fase la de va diseño.”
- **Análisis semántico:** Se les asigna significados a las estructuras creadas por el analizador sintáctico. En otras palabras, se hace una correspondencia entre las estructuras sintácticas y los objetos del dominio de la tarea. Las estructuras en las que no se puede hacer esta correspondencia se rechazan. Por ejemplo, en la mayoría de los universos, “Las ideas verdes incoloras duermen furiosamente” (Chomsky, 1957 [CHO57]) sería desechada por ser semánticamente anómala.
- **Integración del discurso:** Joshi (1981 [JOS81]), el significado de una frase individual puede depender de las frases precedentes y puede influenciar el significado de las frases posteriores. Por ejemplo, la palabra “it” en “John wanted it” (John lo quiso) depende del contexto del discurso, mientras que la palabra “John” puede influenciar el significado de frases posteriores (como “He always had”. (El siempre tuvo).
- **Análisis de la pragmática:** La estructura que representa qué se ha dicho se reinterpreta para determinar su significado actual. Por ejemplo, la frase “Do you know what system is it?” (¿sabe usted qué sistema es?) debería interpretarse como una petición del nombre del sistema.

Los límites entre estas cinco fases con frecuencia son muy difusos. En ocasiones las fases se desarrollan en secuencia, mientras que otras veces se realizan a la vez. Si se realizan en secuencia, una puede pedir ayuda a las demás

Concretamente, para lograr que el problema de la comprensión del lenguaje en su conjunto sea tratable, ayudará el hecho de distinguir entre las siguientes dos formas de descomposición de un programa:

- Los procesos y el conocimiento necesarios para llevar a cabo la tarea.
- La estructura de control global que se impone en estos procesos.

En este tema, nos centraremos de forma especial en el primero de estos aspectos. A partir de considerar un ejemplo para ver el funcionamiento de un proceso concreto. En este ejemplo, se asume que los procesos ocurren secuencialmente. Suponiendo que tenemos una interfaz en inglés para un sistema operativo y se teclea la siguiente oración:

I want to print Bill's *.software* file  
(Quiero imprimir el archivo *.software* de Bill).

### V.3.1. Análisis morfológico

El análisis morfológico debe realizar las siguientes tareas:

- Separar las palabras “Bill’s” en el nombre propio “Bill” y el sufijo posesivo “s”.
- Reconocer la secuencia “.software” como la extensión de un archivo que funciona como un adjetivo en la oración.

Además, el proceso normalmente asignará categorías sintácticas a todas las palabras de la oración. Esto se suele hacer en este momento porque las interpretaciones de los afijos (prefijos y sufijos) suelen depender de la categoría sintáctica de la palabra completa. Por ejemplo, considere la palabra “prints”. Esta palabra es o bien un nombre plural (con la “s” indicando plural) o la tercera persona singular de un verbo (como en “he prints” [él imprime])m en cuyo caso la “s” indica número singular y tercera persona. Si este paso se realiza ahora, en nuestro ejemplo existirá ambigüedad ya que “want”, “print” y “file” pueden todas ellas funcionar con más de una categoría sintáctica.

En el análisis sintáctico debe utilizar los resultados del análisis morfológico para construir una descripción estructurada de la oración. El objetivo de este proceso denominado análisis(*parsing*), es convertir la lista lineal de palabras que constituyen una oración en una estructura que defina las unidades que representan la lista lineal.

### V.3.2. Procesamiento Sintáctico

El procesamiento sintáctico es el paso en el cual una oración lineal de entrada se convierte en una estructura jerárquica que se corresponde con las unidades de significado de la oración Winograd (1983 [WIN83]) y King (1983 [KIN83]). Aunque existen sistemas de comprensión del lenguaje natural que evitan este paso, juega un importante papel en muchos sistemas de comprensión del lenguaje principalmente por dos razones:

- El procesamiento semántico funciona sobre los constituyentes de la oración. Si no existe un paso de análisis sintáctico, el sistema semántico debe identificar sus propios constituyentes. Por otro lado, si se realiza un análisis sintáctico, se restringe el número de constituyentes a considerar por el semántico. El análisis sintáctico es menos costoso, computacionalmente hablando, que el análisis semántico (que requiere inferencias

importantes). Así, pues desempeña un importante papel como reductor de la complejidad global del sistema.

- Aunque frecuentemente se puede extraer el significado de una oración sin usar hechos gramaticales, no siempre es posible hacerlo.

Aunque existen muchas formas de realizar un análisis sintáctico, casi todos los sistemas que se utilizan realmente tienen dos componentes principales:

- Una representación declarativa, denominada gramática, de los hechos sintácticos sobre el lenguaje.
- Un procedimiento, denominado analizador (*parser*), que compara la gramática con las oraciones de entrada para producir estructuras analizadas.

### Gramáticas y analizadores

La forma más usual de representar las gramáticas es mediante un conjunto de reglas de producción. Aunque varían los detalles de la forma permitida por las reglas, la idea básica aparece en todas ellas y se ilustra en la Figura V.10 que muestra una sencilla gramática libre de contexto para representar la estructura de una frase en inglés. La primera regla se lee así, “Una oración se compone de un sintagma nominal seguido por un sintagma verbal”. En esta gramática la barra vertical se lee “o”. El  $\epsilon$  denota la cadena vacía. Los símbolos que son susceptibles de expandirse mediante reglas se denominan símbolos no terminales. Aquellos símbolos que se corresponden directamente con las cadenas que deben encontrarse en la oración de entrada reciben el nombre de símbolos terminales.

O	→	SN SV
SN	→	EI SN1
SN	→	PRO
SN	→	NP
SN	→	SN1
SN1	→	ADJS N
ADJS	→	$\epsilon$   ADJ ADJS
SV	→	V
SV	→	V SN
N	→	PROYECTO   IMPRESORA
SN	→	Tutor
PRO	→	YO
ADJ	→	CORTO   LARGO   RÁPIDO
V	→	IMPRESO   CREADO   QUERER

Figura V.10.- Una sencilla gramática para un subconjunto del español.

Los formalismos gramaticales como éste sirven de base a muchas teorías lingüísticas, las cuales proporcionan a su vez, la base de muchos sistemas de comprensión del lenguaje natural. En este punto debe puntualizarse que existe un acuerdo general en que las gramáticas puras y libres de contexto no son eficaces para describir los lenguajes naturales. Como resultado de todo esto, los sistemas de procesamiento del lenguaje natural tienen

menos en común con los sistemas de procesamiento de lenguajes de programación de lo que cabría esperar.

Sin tener en cuenta la base teórica de la gramática, el proceso de análisis sintáctico toma reglas de la gramática y las compara con la oración de entrada. Cada regla que se empareje añade algo a la estructura de la oración que se está construyendo. La estructura más simple que se puede construir es un árbol de análisis (*parse tree*), en el que simplemente se almacenan las reglas y la forma en que se han emparejado. La Figura V.11, muestra el árbol de análisis obtenido de la oración “*Tutor imprimió el proyecto*”, usando la gramática definida.

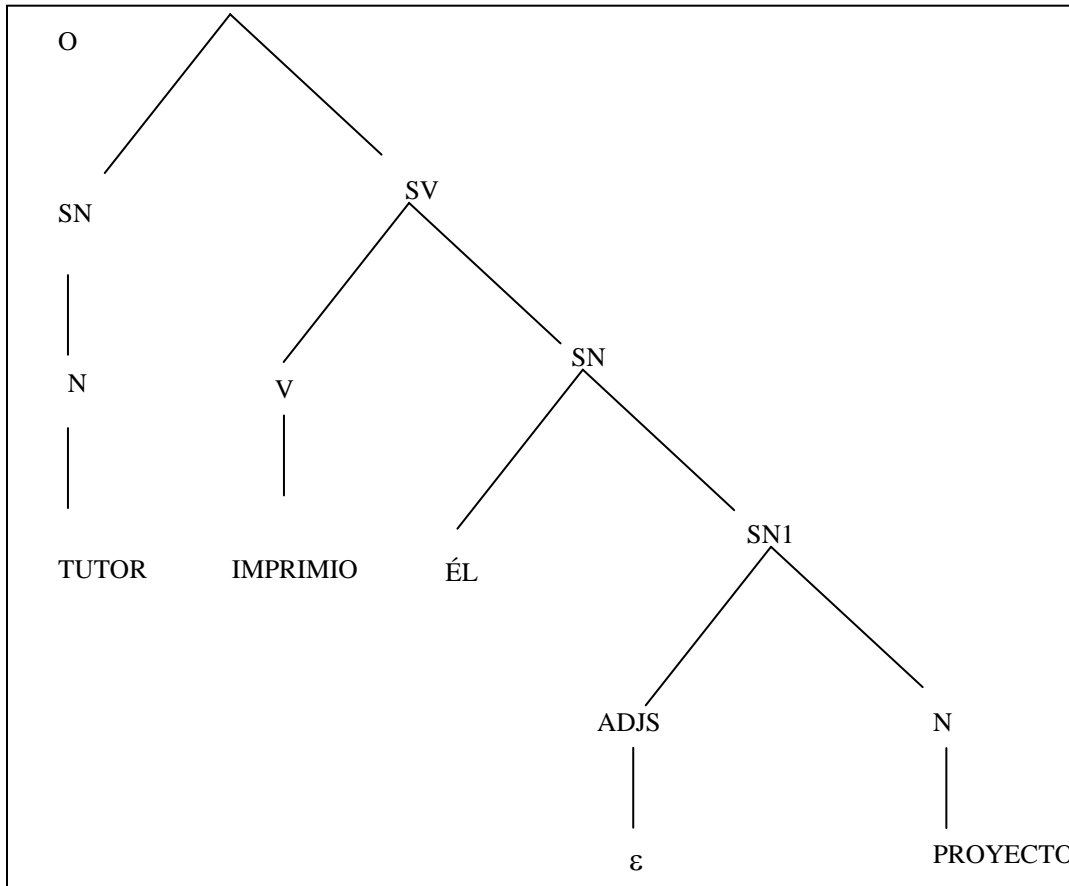


Figura V.11.- El árbol de análisis de una frase.

Nótese que cada nodo del árbol de análisis se corresponde con una palabra de la entrada o con un símbolo no terminal de nuestra gramática. Cada nivel del árbol de análisis se corresponde con la aplicación de una regla gramatical. Como resultado, es claro ver que una gramática específica dos cosas sobre el lenguaje:

- Su pobre capacidad generativa, por lo cual entendemos el conjunto de las oraciones contenidas en el lenguaje. Este conjunto (denominado conjunto de oraciones gramaticales) se construye precisamente con aquellas oraciones que se pueden emparejar completamente por una serie de reglas de la gramática.
- Su fuerte capacidad generativa, por lo cual entendemos las estructuras (o posiblemente estructuras) que se asignan a cada oración gramatical del lenguaje.



### V.3.3. Análisis Semántico

La realización de un análisis sintáctico a una oración es sólo un primer paso para comprenderla. Todavía tenemos que fabricar una representación del significado de la oración (Slocum (1988 [SLO88]), Nirenburg (1987 [NIR87]), Lehrberger y Bourbeau (1988 [LEB88]), y Nagao (1989 [NAG89])). Como la comprensión es una correspondencia, primero tenemos que definir el lenguaje al que estamos intentando llegar. No existe un único y definitivo lenguaje en el que se pueda describir el significado de todas las oraciones.

La elección de un lenguaje destino para un programa concreto de comprensión del lenguaje natural puede depender de lo que se vaya a hacer con los significados una vez que se hayan construido.

Cuando el lenguaje natural se considera un problema en sí mismo, como por ejemplo, cuando se construye un programa cuyo objetivo sea leer texto y responder a preguntas sobre él, el lenguaje destino debe diseñarse de forma específica para que soporte el procesamiento de lenguaje. En este caso, normalmente se buscan primitivas que se correspondan con las distinciones que anormalmente se hacen en el lenguaje. Por supuesto, seleccionar un conjunto apropiado de primitivas no es una tarea sencilla.

Cuando el lenguaje natural se usa como un lenguaje de interfaz con otro programa (como un sistema de peticiones a una base de datos o un sistema experto), el lenguaje destino debe ser una entrada para el otro programa. Así, el diseño del lenguaje destino está influido por este programa, en este caso es útil usar una representación intermedia basada en conocimiento para guiar el proceso en su conjunto. Así, en el resto de este apartado se asume que el lenguaje destino que estamos construyendo está basado en conocimiento.

Aunque el propósito principal del procesamiento semántico es la creación de una representación en el lenguaje destino del significado de una oración juega también otro papel importante. Impone restricciones a las representaciones que se pueden construir y, debido a las conexiones estructurales que deben existir entre la estructura sintáctica y semántica, también proporciona una forma de selección entre distintos análisis sintácticos que compiten entre sí. El procesamiento semántico puede imponer restricciones porque tiene acceso al conocimiento sobre el sentido del mundo.

#### Gramáticas Semánticas

Una gramática semántica (Burton, 1976 [BUR76], Hendrix y Lewis, 1981 [HEL81]) es una gramática libre de contexto en donde la elección de los no terminales y las reglas de producción se gobierna mediante una función tanto semántica como sintáctica. Además, existe normalmente una acción semántica asociada a cada regla de la gramática. El resultado del análisis y la aplicación de todas las acciones semánticas asociadas, es el significado de la oración. Este fuerte emparejamiento entre las acciones semánticas y las reglas de la gramática funciona porque las reglas de la gramática están diseñadas teniendo en cuenta conceptos semánticos claves.

En la Figura V.12 se muestra un ejemplo de un fragmento de una gramática semántica. Esta gramática define parte de una sencilla interfaz para un sistema operativo. Debajo de cada regla aparece entre llaves la acción semántica que se realiza cuando se aplica. El término "valor" se utiliza para referirse al valor que se empareja con la parte derecha de la regla. La notación del punto empleada x.y significa que y es un atributo de la unidad x. El resultado de un análisis realizado con éxito usando esta gramática será o una orden o una petición.

O → Cual es la PROPIEDAD DE ARCHIVO del ARCHIVO?  
 {consulta ARCHIVO PROPIEDAD DE ARCHIVO}

O → Yo quiero la ACCION  
 {ordenar ACCION}

PROPIEDAD DE ARCHIVO → the PROP. ARCHIVO  
 {PROP.ARCHIVO}

PROP. ARCHIVO → extensión | protección | fecha de creación | usuario  
 {valor}

ARCHIVO → NOMBRE ARCHIVO | archivo1  
 {valor}

ARCHIVO1 → PROCEDIMIENTO's ARCHIVO2  
 {ARCHIVO2 propietario: USUARIO}

ARCHIVO1 → ARCHIVO2  
 (ARCHIVO2)

ARCHIVO2 → EXT file  
 {instancia: estructura de archivo  
 extensión: EXT}

EXT → . init |.bd | .isp | .for | .ps | .mss  
 {valor}

ACCION → imprimir ARCHIVO  
 {instancia: imprimir  
 objeto: ARCHIVO}

ACCION → imprimir ARCHIVO en IMPRESORA  
 {instancia: imprimir  
 objeto: ARCHIVO  
 impresora: IMPRESORA}

PROCEDIMIENTO → Metodologia | Proyecto  
 {valor}

Figura V.12.- Una gramática semántica

La Figura V.13 muestra el resultado de aplicar esta gramática semántica a la oración:  
*“Tutor quiere imprimir el archivo .init del Proyecto”*

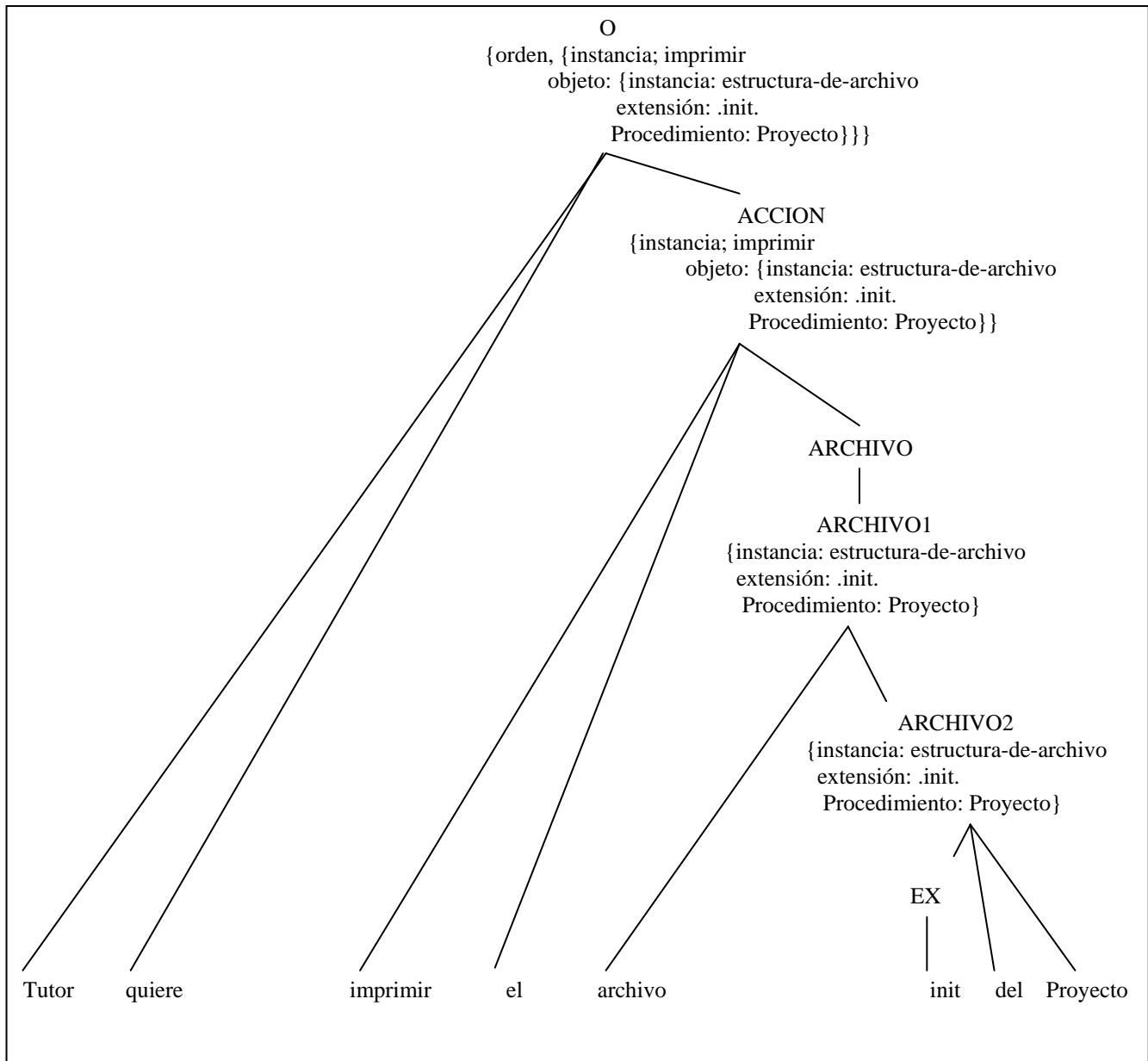


Figura V.13.- El resultado de análisis con una gramática semántica.

Este enfoque combina todo el procesamiento de LN, con la excepción de la parte final del procesamiento de la pragmática en el que se realiza la conversión a la sintaxis de las órdenes del sistema. La conclusión a la que se ha llegado es que pueden ser muy útiles para producir rápidamente interfaces de lenguaje natural restringido. Pero como solución total al problema de la comprensión del Lenguaje Natural, no pueden funcionar debido a su inoperancia para capturar generalizaciones lingüísticas importantes.

## Capítulo VI. Diseño del PROTOTIPO SE-APIS del Sistema Basado en Conocimiento

El Modelo del PROTOTIPO de la Arquitectura del Sistema Basado en Conocimiento SE-APIS debe cumplir con los objetivos y alcances definidos en el capítulo I, considerando el dominio de conocimientos definidos en el capítulo III y permitiendo almacenar en su base de conocimientos las instancias del CASO de la Metodología de desarrollo de aplicaciones CASE (CADM) descrito en el capítulo IV. Las técnicas de la Inteligencia Artificial que se utilizaran para el desarrollo del modelo y su construcción son las planteadas en el capítulo V, que consisten fundamentalmente de: *Marcos* de Minsky, Dependencias Conceptuales de Shanck, y procesamiento de LN.

La Arquitectura del SE-APIS debe cumplir con las siguientes especificaciones de procesamiento:

### VI.1. Especificaciones de Procesamiento

- a) El Modelo de la Arquitectura del SE-APIS debe ser enfocado a estructurar una base de conocimientos que sirva para procesar metodologías de desarrollo de sistemas de información, estructuras para Administración de Proyectos de Ingeniería de *Software*, y el modelo de Putnam que muestra la representación de conocimientos de métricas para desarrollo de proyectos. El modelo de esta Arquitectura PROTOTIPO debe estar considerada como parte de un sistema experto integral para apoyo a la Ingeniería de *Software*.
- b) Diseño y Construcción de una base de conocimientos estructurada con ranuras de relleno débil *Marcos*, que permiten establecer Superclases y Clases de Objetos entre los diferentes niveles del conocimiento relacionados con atributos **is-a (es un)**, que facilita aplicar de manera natural un Lenguaje Orientado a Objetos. El modelo de la estructura de esta base de conocimientos debe ser fácilmente transportada a lenguajes de programación que permitan POO (Programación Orientada a Objetos) y el modelo de las instancias de los *Marcos* deben ser fácilmente reusadas en una BDOO (Base de Datos Orientada a Objetos).
- c) Estructurar en forma modular la base de conocimientos de *Marcos* del SE-APIS para que en un trabajo futuro sea ampliada en los niveles quinto y sexto para soportar una interfaz con Modelos y Métodos de análisis y diseño de sistemas, tales como BDD E-R (Base de Datos Entidad-Relación) y diagramas flujo de procesos.
- d) Construcción de reglas de inferencia utilizando estructuras de relleno fuertes particularmente dependencias conceptuales (**CD**) que tradicionalmente se han utilizado en aplicaciones de Inteligencia Artificial de robótica para sujetos y objetos en movimiento dentro de un espacio. En este sentido en el SE-APIS se aplican de manera original las dependencias conceptuales en el manejo de información estructurada.
- e) Desarrollo de un consultor computarizado de metodologías de Ingeniería de *Software*, incorporando en la base de conocimientos del SE-APIS, metodologías de esta especialidad. Almacenando como una de las instancias de prueba de manera particular en la base de

conocimientos la Metodología de Desarrollo de Aplicaciones CASE (CADM) de la compañía ORACLE.

f) Definición de las **CD** de consulta() y reporta(), para incorporarlas en las reglas de inferencia programadas dentro de un motor de inferencia con control *oportunistico* con encadenamiento hacia adelante, el cual genera la respuesta requerida por el usuario de acuerdo con sus requerimientos de los diferentes niveles de la base de conocimientos.

g) Generación automática de Proyectos de Ingeniería de *Software* estándar para desarrollo de Sistemas de Información, incorporando en la base de conocimientos valores ponderados de acuerdo a las curvas paramétricas del comportamiento de proyectos en cada una de las etapas y actividades de la metodología CADM y vinculándolas con estructuras para Administración de Proyectos: WBS (*Work BreakDown Structure*) y PDM (*Precedence Diagramming Method*) por medio de *Marcos* de hechos y reglas de conocimiento.

h) Desarrollo de las reglas de inferencia para calcular el estimado de la cantidad de líneas de código del *software* a realizar a partir del estimado de horas de esfuerzo total que el proyecto requerirá para desarrollo, por medio de las métricas del modelo de Putnam.

i) Construcción de la **CD** crea() para la generación automática de Proyectos de Ingeniería de *Software* y del cálculo de la métrica del modelo de Putnam, a partir de la base de conocimientos de Metodologías, Proyectos y de reglas de inferencia.

j) Desarrollo del modelo del cálculo de la ruta crítica utilizando una red de etapas y un proceso inteligente de red reducida de actividades, en lugar del método tradicional.

k) Diseño de la dependencia conceptual (**CD**) programa() para el cálculo de la ruta crítica.

l) Desarrollo de las reglas de inferencia y control dentro del motor de inferencia para soportar las metas de las **CD** consulta(), reporta(), programa() y crea().

m) Diseño de los *Marcos* de Modelos y Métodos relacionados en el quinto y sexto nivel de la base de conocimientos para establecer una vinculación futura con herramientas de *software* inteligente o CASE para procesamiento de Sistemas de Información.

n) Documentación de la definición de las reglas de las **CD** para promover en futuros trabajos de tesis la ampliación de la gama de **CD** para completar el proceso del SE-APIS en sus acciones de controlar(), cerrar() y evaluar(), analizar(), diseñar() en la construcción de Sistemas de Información.

o) Desarrollar una interfaz con el usuario, empleando frases de Lenguaje Natural (**LN**) soportadas por una gramática sencilla con estructura específica, construida con reglas de inferencia para la interpretación de oraciones por medio análisis morfológico, sintáctico y semántico que generan en forma dinámica la estructura de las **CD** del SE-APIS.

p) Construcción de la programación del PROTOTIPO del SE-APIS con la Herramienta para desarrollo de sistemas expertos *CLIPS* (C Lenguaje Integrated Production System) versión 6.05 de *Software Technology Branch* (STB) de la NASA, el cual soporta de forma amplia los objetivos específicos listados.

## VI.2. Modelo de *Marcos de Conocimiento del SE-APIS*

Los modelos son usados como un medio de comunicación entre el EH (experto humano) y los Ingenieros del Conocimiento como una guía en la estructuración de la arquitectura del sistema experto independientemente del *software* que se utilizará para su implantación. Así mismo el diseño de modelos facilita la adquisición de conocimientos, identificar fuentes de conocimiento, representar los conocimientos y finalmente permiten evaluar el impacto positivo o negativo que tendrá el sistema experto.

El modelo de *marcos* de conocimiento (objetos) del PROTOTIPO del sistema experto de apoyo a la Administración de Proyectos de Ingeniería de *Software* SE-APIS, está formado por diferentes niveles de clases y superclases, integrado por dos vistas: la estructura de Metodologías y la estructura de Proyectos diseñadas para ser soportadas y programadas en el lenguaje **CLIPS (C Language Integrated Production System)** de la NASA.

### VI.2.1. Modelo de las Metodologías de Ingeniería de *Software*

De acuerdo con la Biblioteca de CASOS definidos en el capítulo IV, a partir de la metodología **CADM**, se pueden abstraer tres *marcos* de conocimiento: Metodología(s), Fase(s) y Actividades.

En el diseño de la base de conocimientos podemos suponer que una metodología está formada de una o más fases, y que una fase puede estar formada por una o más actividades. Por lo tanto, podemos definir que **Actividades** es un subconjunto de **Fases**, y que a su vez **Fases** es un subconjunto de **Metodologías**.

Adicionalmente también podemos definir que una metodología forma parte de los procedimientos que se realizan en una organización, por lo que se puede expresar que el marco **Metodologías** es un subconjunto del marco **Procedimientos** y Procedimientos es un subconjunto de la Organización.

De la misma manera si analizamos el contenido de las Actividad(es) podemos identificar que para llevar a cabo algunas de ellas es necesario realizar algún tipo de modelo (conceptual, lógico o físico) y que para realizar un modelo es necesario aplicar uno o varios métodos para desarrollo de *software*, por ejemplo: Base de Datos Entidad Relación (E-R), diagramas de proceso del método Yourdon, estructuración modular, diagramas lógicos de programación, etc., contenidas en la clase de sexto nivel Método(s) la cual a su vez es una subclase de la clase Modelo(s) y a su vez Modelo(s) es subclase de Actividad(es).

Al aplicar la relación entre subconjuntos **is-a** a los *marcos* Procedimiento(s), Metodología(s), Fase(s), Actividad(es), Modelo(s) y Métodos obtenemos un diagrama de relación entre clases con una herencia jerárquica simple, en donde cada clase tiene como máximo una sola superclase y que se muestra en la Figura VI.1. El marco Organización en este trabajo no se utiliza pero es posible utilizarlo como metaclass.

En un sentido estricto las relaciones entre clases de metodologías deben ser **part-of** en lugar de **is-a**, dado que por ejemplo una fase es parte de una metodología, sin embargo, el lenguaje de programación *CLIPS* no tiene soporte para la primera de estas relaciones y cuando se programa con la segunda de ellas proporciona resultados consistentes.

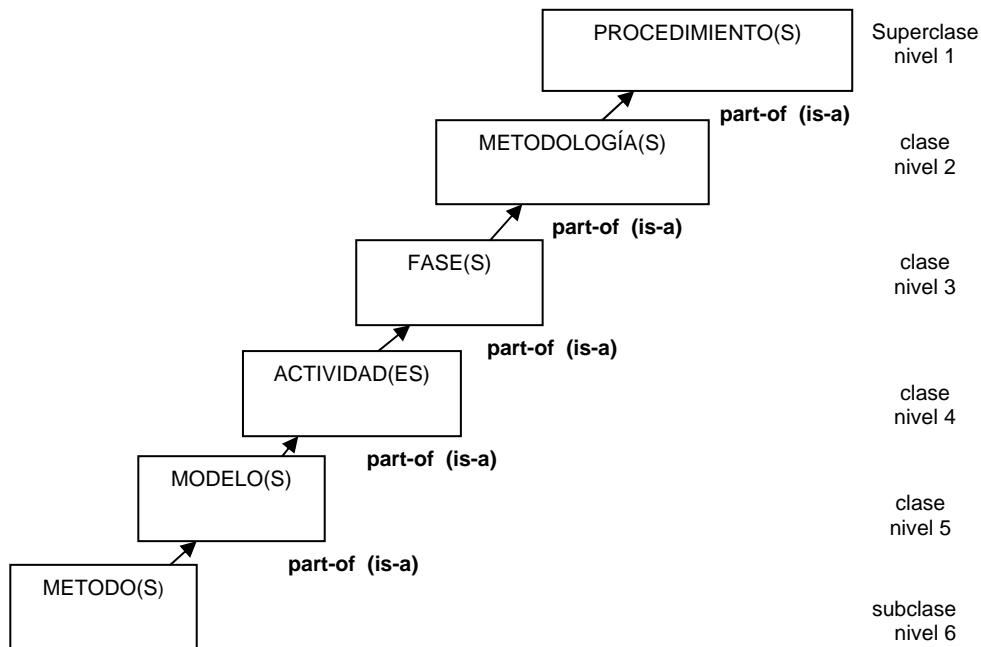


Figura VI.1.- Modelo de relaciones entre clases de Metodologías de Ingeniería de *Software*.

El modelo de relaciones entre clases de metodologías de ingeniería de *software* utilizando relaciones *is-a* y *part-of* de la forma siguiente:

<b>PROCEDIMIENTOS</b>	( <i>is-a</i> USER)
<b>METODOLOGÍAS</b>	( <i>is-a</i> PROCEDIMIENTOS)
<b>FASES</b>	( <i>part-of</i> METODOLOGÍAS)
<b>ACTIVIDADES</b>	( <i>part-of</i> FASES)
<b>MODELOS</b>	( <i>part-of</i> ACTIVIDADES)
<b>MÉTODOS</b>	( <i>part-of</i> MODELOS)

Considerando métodos de representación del conocimiento de estructuras de ranura y relleno débiles es posible asociar a los *marcos* (objetos) sus atributos representados como ranuras (*slots*), que de acuerdo con información básica de una metodología son los siguientes:

PROCEDIMIENTOS  
slot procedimiento  
slot objetivo

METODOLOGÍAS  
slot metodología  
slot id\_metodología  
slot alcance

FASES  
slot etapa  
slot id\_etapa  
slot descripción\_etapa  
slot peso\_etapa

ACTIVIDADES  
slot actividad  
slot id\_actividad  
slot peso\_actividad  
slot trabajo  
slot resp\_actividad  
slot precedencias

MODELOS  
slot modelo  
slot id\_modelo  
slot descripción\_modelo

MÉTODOS  
slot método  
slot id\_método  
slot descripción\_método  
slot herramientas\_CASE  
slot simbología

## VI.2.2. Modelo de Administración de Proyectos

Como parte de la Arquitectura del modelo de la base conocimientos del sistema experto APIS, es necesario definir el modelo de la Administración de Proyectos de acuerdo con los conceptos descritos en el capítulo III correspondiente al dominio del SE-APIS.

Un proyecto de Ingeniería de *Software* necesariamente debe considerar un encabezado en donde se almacenen su identificación y sus datos globales y adicionalmente para estructurar el alcance definido por el usuario se debe definir con una **WBS (Work BreakDown Structure)** adecuada para realizar la fragmentación de trabajo requerido.

La estructura de la **WBS** para ingeniería de *software* tiene como primer nivel la identificación del *software* o sistema de información, como segundo nivel las fases del ciclo de vida del *software* (planeación, análisis, diseño, construcción, pruebas, implementación y mantenimiento) y como tercer nivel las operaciones para realizar los paquetes de trabajo del sistema, estos niveles se muestran en la Figura VI.2.

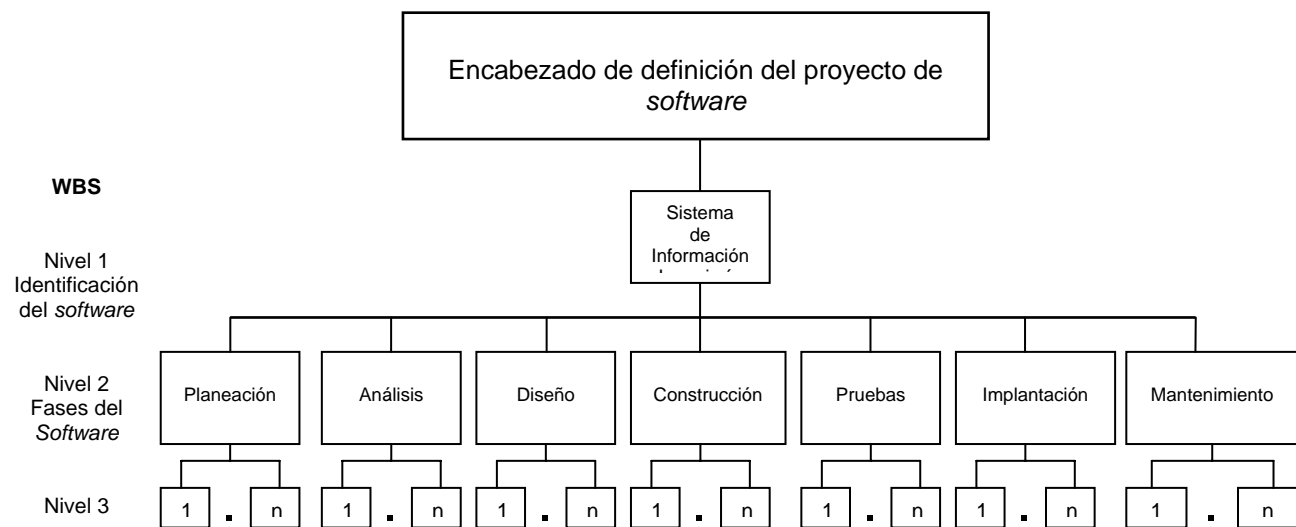


Figura VI.2 Definición y WBS de un proyecto de Ingeniería de *Software*

De acuerdo con esta estructura de información podemos identificar los siguientes *marcos*: **Proyecto(s)** (definición del proyecto), **WBS(s)** de primer y segundo nivel (identificación de proyectos y fases de *software*) y **Operación(es)**, así como los **Recurso(s)** humanos que serán los responsables de realizar cada una de las operaciones. Adicionalmente debemos de considerar que el marco Proyecto(s) es un subconjunto del marco **Procedimientos** que se definió anteriormente.

Al aplicar la relación entre subconjuntos **is-a** a los *marcos* Procedimientos, Proyectos(s), WBS(s), Actividad(es), Operaciones(s) y Recurso(s) obtenemos un diagrama de relación entre clases con una herencia jerárquica simple, en donde cada clase tiene como máximo una sola superclase y que se muestra en la Figura VI.3.

Como ya se mencionó anteriormente las relaciones adecuadas entre los subconjuntos de los *marcos* para este modelo deben ser **part.-of**, sin embargo *CLIPS* solo soporta **is-a**, sin provocar problemas de consistencia.



Antes de establecer las relaciones de clases entre *marcos* y de expresarlo en el lenguaje de programación *CLIPS*, es muy importante tomar en cuenta que los WBS(s), las Operaciones y los Recurso(s) humanos en un proyecto real de ingeniería de *software* pueden variar y no ser exactamente las que una metodología estándar propone, de tal forma que al representarlas en la base de conocimientos no se expresan como una clase o marco, sino como una estructura plana de datos relacional *template* (similar a las tablas de las bases de datos E-R), esto con el fin de permitir que en el *software CLIPS* puedan crecer las fases de un proyecto en un WBS(s) de manera dinámica.

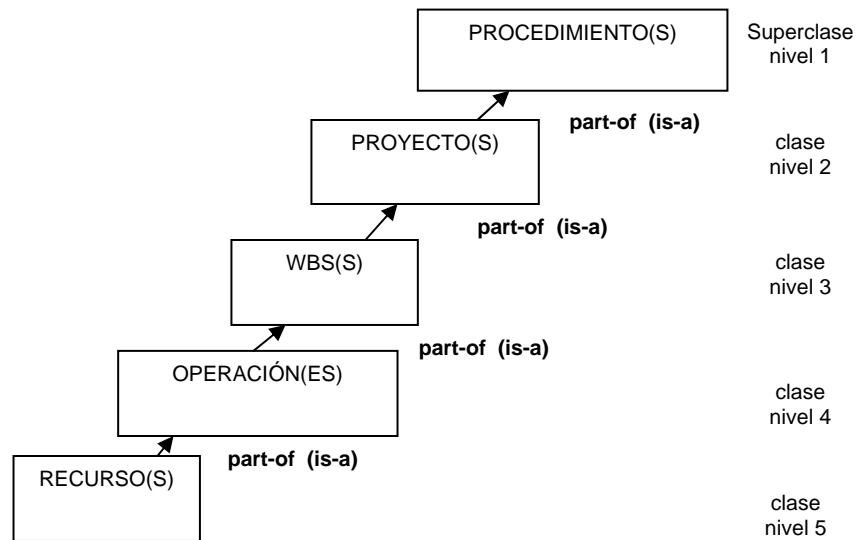


Figura VI.3.- Modelo de relaciones entre clases de Proyectos de ingeniería de *software*.

La representación de la vista de administración de proyectos de ingeniería de *software* como *clases* de conocimiento relacional de proyectos de ingeniería de *software* se expresa de la siguiente forma:

<b>PROCEDIMIENTOS</b>	<b>(is-a USER)</b>
<b>PROYECTOS</b>	<b>(is-a PROCEDIMIENTOS)</b>
<b>WBSS</b>	<b>(part-of PROYECTOS)</b>
<b>OPERACIONES</b>	<b>(part-of WBSS)</b>
<b>RECURSOS</b>	<b>(part-of OPERACIONES)</b>

Por ser instancias WBSS, Operaciones y Recursos, en una representación real de conocimiento en *CLIPS* no es posible aprovechar la propiedad de herencia de atributos de las clases a las subclases que el lenguaje de programación orientado a objetos facilita, por lo que tendrán que ser transferidos por medio de reglas de control que establezcan relaciones entre pares de atributos de los *templates*.

Para poder representar en la base de conocimiento la administración de proyectos de *software* es necesario considerar que dentro de los atributos básicos en la planeación y ejecución de las operaciones de un proyecto de *software* deben incluirse el costo y tiempo de duración para su desarrollo así como la asignación de los recursos humanos que forman el

grupo de trabajo responsable, quedando los *marcos* como estructuras de ranura y relleno débiles (*slots*) definidos con sus atributos como sigue:

#### PROCEDIMIENTOS

slot procedimiento  
slot objetivo

#### PROYECTOS

slot proyecto  
slot id\_proyecto  
slot objetivo  
slot alcance  
slot fecha\_inicio  
slot fecha\_final  
slot dur\_proy  
slot trabajo  
slot costo  
slot resp\_proy

#### WBSS

slot proyecto  
multislot wbs  
slot id\_wbs  
slot fecha\_inicio\_wbs  
slot fecha\_final\_wbs  
slot dur\_wbs  
slot trabajo\_wbs  
slot costo\_wbs  
slot resp\_etapa\_wbs

#### OPERACIONES

slot proyecto  
slot id\_wbs  
slot operación  
slot id\_operación  
slot dur\_operación  
slot trabajo\_oper  
slot costo\_oper  
slot resp\_oper  
multislot precedencias

#### RECURSOS

slot recurso  
slot proyecto  
slot id\_wbs  
slot id\_operación  
slot id\_recurso  
slot trabajo  
slot costo\_rec  
slot responsabilidad

Cuando en el sistema experto APIS se crea un proyecto, el marco **WBSS** se genera a partir del marco **FASES** de la base de conocimientos de tal forma que el número de *instancias* de los dos *marcos* son iguales en la primera planeación del proyecto que se genera de manera automática, de tal forma que el *slot* etapa es igual al *slot* de wbs. Esto significa que inicialmente las WBS(s) son un subconjunto de las FASES (WBSS **is-a** FASES), aunque posteriormente el número de WBS(s) puede ser mayor o menor de acuerdo con la modificación de la planeación que el administrador del proyecto pueda realizar.

De la misma manera al crear el proyecto las **OPERACIONES** de un proyecto se generan a partir de las **ACTIVIDADES**, de tal forma que el *slot* operación es igual al *slot* actividad, siendo las OPERACIONES un subconjunto de las ACTIVIDADES (OPERACIONES **is-a** ACTIVIDADES) y su número *instancias* inicialmente es el mismo, el cual puede posteriormente ser modificado en una segunda programación por el administrador del proyecto.

En condiciones iniciales de creación automática de un proyecto, el marco **MODELOS** que forma parte o es un subconjunto (**part-of** o **is-a**) del marco **ACTIVIDADES**, también es un subconjunto del marco **OPERACIONES**.

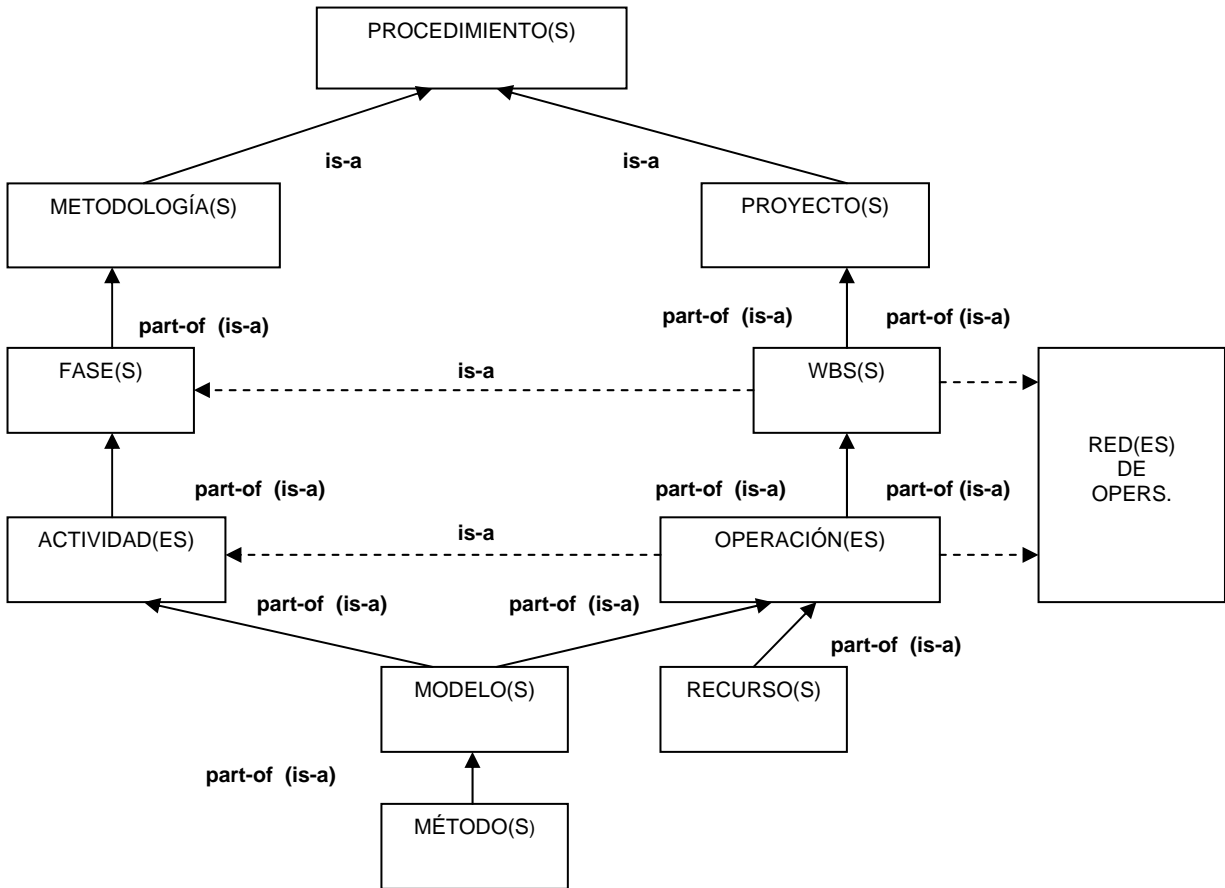


Figura VI.4.- Modelo completo de relaciones entre clases de ingeniería de *software*.

Las fases **WBSS** y las **OPERACIONES** se ligan por medio de precedencias entre sí para formar el marco **REDES** de operaciones del proyecto. Estas **REDES** de fases y operaciones requieren de estructuras *templates* para almacenar temporalmente los datos de las relaciones de precedencias de la red del proyecto de ingeniería de *software*, el *template* para almacenar el resultado de aplicar el método del cálculo de la ruta crítica a la red de fases y operaciones, así como una estructura *template* para almacenar la ruta más larga temporal durante las iteraciones del algoritmo, el procedimiento y las reglas de control para este cálculo se explicaran más adelante en la definición de los procesos del sistema experto.

El modelo completo de la base de conocimientos de Ingeniería de *Software* se representa en el diagrama de la Figura VI.4 y considera la estructura de los *marcos* de la vista del modelo de relaciones entre clases de Metodologías de ingeniería de *software* y la vista del modelo de relaciones entre clases de Proyectos de ingeniería de *software*. Adicionalmente considera las relaciones mencionadas entre FASES y WBSS, ACTIVIDADES y OPERACIONES, MODELOS y el marco REDES formado por estructuras de datos *templates*.

La representación de los *marcos*, las relaciones entre clases y las estructuras *templates* del modelo de la base de conocimientos de ingeniería de *software* como conocimiento relacional se expresa de la siguiente forma:

<b>PROCEDIMIENTOS</b>	(is-a USER)
<b>METODOLOGÍAS</b>	(is-a PROCEDIMIENTOS)
<b>FASES</b>	(part-of METODOLOGÍAS)
<b>ACTIVIDADES</b>	(part-of FASES)
<b>MODELOS</b>	(part-of ACTIVIDADES operaciones)
<b>MÉTODOS</b>	(part-of MODELOS)
<b>PROYECTOS</b>	(is-a PROCEDIMIENTOS)
<b>WBSS</b>	(part-of PROYECTOS)
<b>OPERACIONES</b>	(part-of WBSS)
<b>REDES</b>	(part-of OPERACIONES)
<b>RECURSOS</b>	(part-of OPERACIONES)

La relación **is-a** para los *templates* no son declarados explícitamente en *CLIPS*, sino que a partir de un par de atributos relacionados y con apoyo de reglas condicionales del tipo **if.then** se establece la relación de clase y la herencia de atributos se realiza mediante la creación de hechos que son asignados al *template* que forma la subclase. El tratamiento de esta forma le da mayor flexibilidad al sistema experto para que puedan crearse tantas etapas de **WBSS** y **OPERACIONES** adicionales del proyecto de ingeniería de *software* como el administrador del mismo decida.

### VI.3. Modelo de Procesamiento del SE-APIS

El modelo de procesamiento del sistema experto **APIS** (Herramienta de Ingeniería de *Software*), tiene una interfaz con el usuario que permite interactuar con el sistema por medio de frases de *Lenguaje Natural* que son procesadas por el motor de inferencia *CLIPS*. Estas frases o declarativas son analizadas por las reglas de inferencia programadas en *CLIPS* que forman el analizador morfológico y el analizador sintáctico para evaluar que las frases estén bien estructuradas con un significado comprensible y lógico.

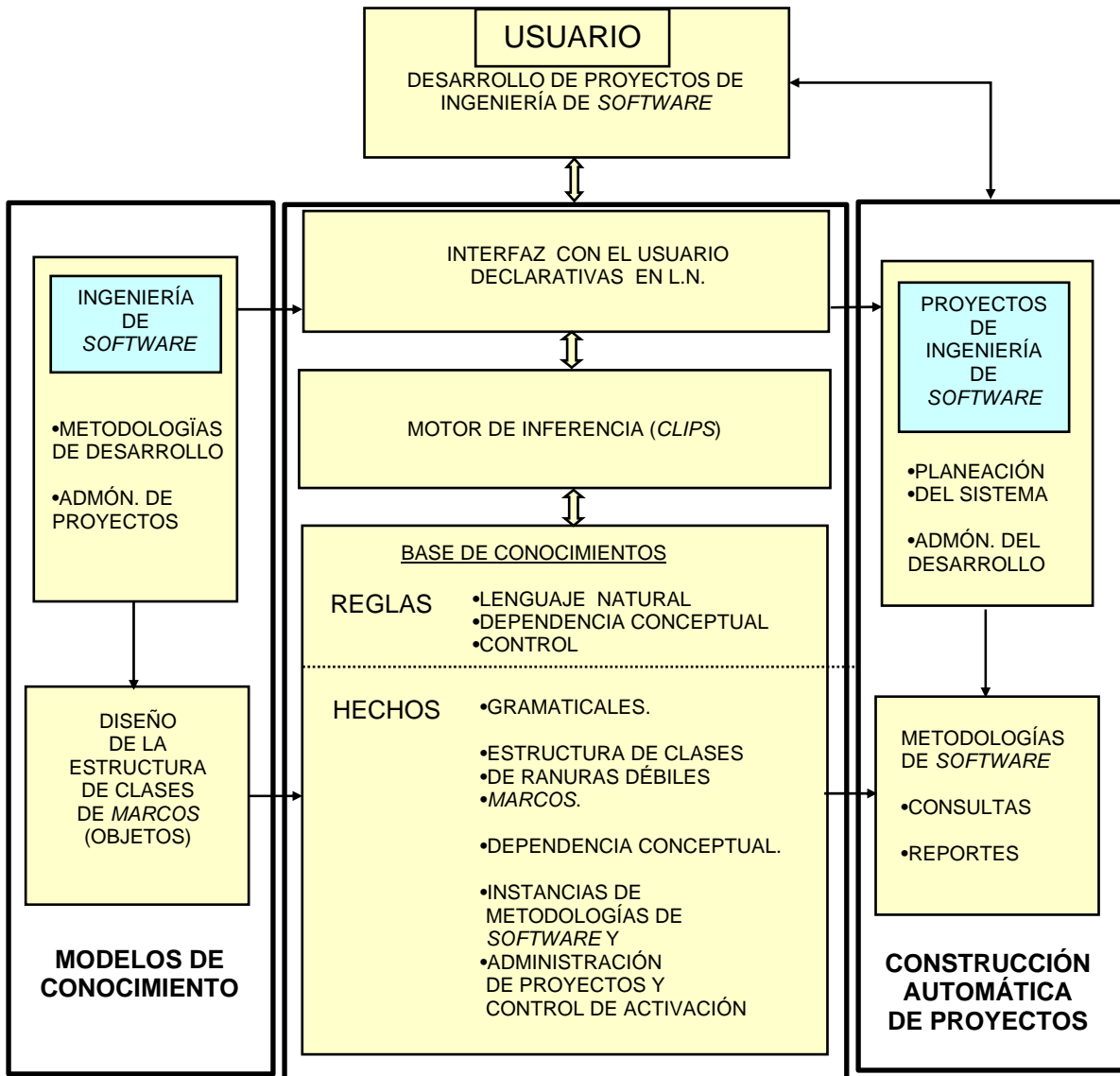


Figura VI.5.- Modelo de procesamiento del sistema experto APIS.

El resultado del procesamiento del análisis morfológico y sintáctico permite obtener los componentes gramaticales: verbo, sujeto (actor), clase origen (y destino en su caso), y las instancias de información, lo cual es analizado por las reglas de inferencia programadas del procesador semántico para llenar la estructura de las dependencias conceptuales **CD**.

Dentro de las reglas de inferencia de procesamiento de conocimiento se realizan comparaciones de los hechos de las estructuras de las dependencias conceptuales **CD** contra las declaraciones de las estructuras de **Marcos** representadas en el lenguaje COOL (Lenguaje orientada a objetos de *CLIPS*, Joseph Giarratano 1997 [CLI97]). Esta comparación activa al motor de inferencia para navegar por la base de hechos (orientada a objetos) de clases, subclases y tablas de relación (*templates*).

Al cumplirse las condiciones de comparación de los hechos (instancias) de ambas estructuras las reglas de inferencia se activan *oportunistamente* y producen los hechos resultantes de acuerdo con las declarativas que están en la parte consecuente de la regla.

Los hechos resultantes que son generados por las reglas de inferencia del sistema experto APIS son procesados en el motor de inferencia, produciendo nuevos hechos de conocimiento que se clasifican de la siguiente forma:

- Hechos de Control.- Incluidos en reglas que controlan el flujo lógico de proceso del sistema experto dentro de la máquina de inferencia *CLIPS*.
- Hechos de Información.- Proporcionan información estructurada de metodologías de Ingeniería de *Software* al usuario, de acuerdo con las clases de *marcos* definidos.
- Hechos de generación automática.- Incluidos en reglas dirigen a la máquina de inferencia en la construcción de la generación automática de Proyectos de Ingeniería de *Software* a partir de las instancias contenidas en la estructura de clases de *marcos* de metodologías de *software*.
- Hechos de Funciones.- Dirigen al motor de inferencia en la ejecución de funciones soportadas por el sistema experto.

### VI.3.1. Procesamiento de Lenguaje Natural del SE-APIS

Para soportar el procesamiento del *Lenguaje Natural* dentro del sistema experto APIS, se ha diseñado una gramática libre de contexto, que acepta oraciones para un lenguaje de expresiones lingüísticas limitado. Específicamente la gramática del APIS procesa oraciones con la estructura de los siguientes ejemplos:

- *Agente consulta el procedimiento de metodologías*
- *Tutor consulta la metodología CADM*
- *Analista reporta la etapa de diseño*
- *Ingeniero crea un proyecto de metodología CADM*
- *Ingeniero programa el proyecto P.00001*
- ...

Estas oraciones están construidas de tal forma que obedecen a las dos estructuras gramaticales específicas siguientes:

***SN SV artículo Clase preposición & Instancia.***

***SN SV artículo Clased preposición Clasef Instancia.***

Donde: SN es sintagma nominal (sujeto).  
SV es sintagma verbal (verbo).

El complemento de la oración está formado por:

- Artículo Artículo.
- Clase Clase de conocimiento.
- Instancia Instancia de conocimiento.
- Clased Clase destino.
- Preposición Preposición.
- Clasef Clase fuente.
- $\epsilon$  cadena vacía.

### Analizador morfológico del APIS.

Dentro del analizador morfológico del APIS se consideran un conjunto de hechos y reglas de conocimiento para poder llevar a cabo la separación y reconocimiento de la secuencia de las palabras permitidas en las oraciones y asignarles una de las siguientes tres categorías sintácticas: sintagma nominal (sujeto), sintagma verbal y complemento.

El dominio de los hechos de conocimiento de las palabras permitidas en las oraciones del APIS son los siguientes:

- SN (sujeto) -> (Agente Tutor Analista Ingeniero)*
- SV(verbo) -> (consulta reporta crea programa controla cierra evalúa)*
- Artículo -> (la el los las un una unos unas)*
- Preposición -> (de a con por)*
- Clase -> (procedimiento metodología etapa actividades proyecto)*
- Clased Clasef -> Clase*
- Instancia -> (administración metodologías CADM objetos prototipos descripciones(etapa) número de proyecto)*

La secuencia de procesamiento de las reglas de inferencia del APIS para realizar el análisis morfológico incluyendo la separación de palabras y la asignación de categorías sintácticas siguen el orden:

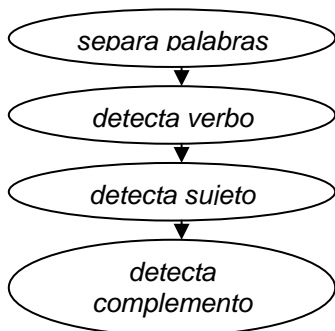


Figura VI.6.- Procesamiento del análisis morfológico del SE-APIS

## Analizador sintáctico del APIS.

El análisis sintáctico en el APIS aprovecha los resultados obtenidos del análisis morfológico y prácticamente se inicia cuando se hace la asignación de las categorías sintácticas: sujeto, verbo y complemento. Adicionalmente para completar el análisis sintáctico es necesario definir un conjunto de reglas de producción de una gramática libre de contexto específica que soporte las oraciones en lenguaje natural de la interfaz con el usuario.

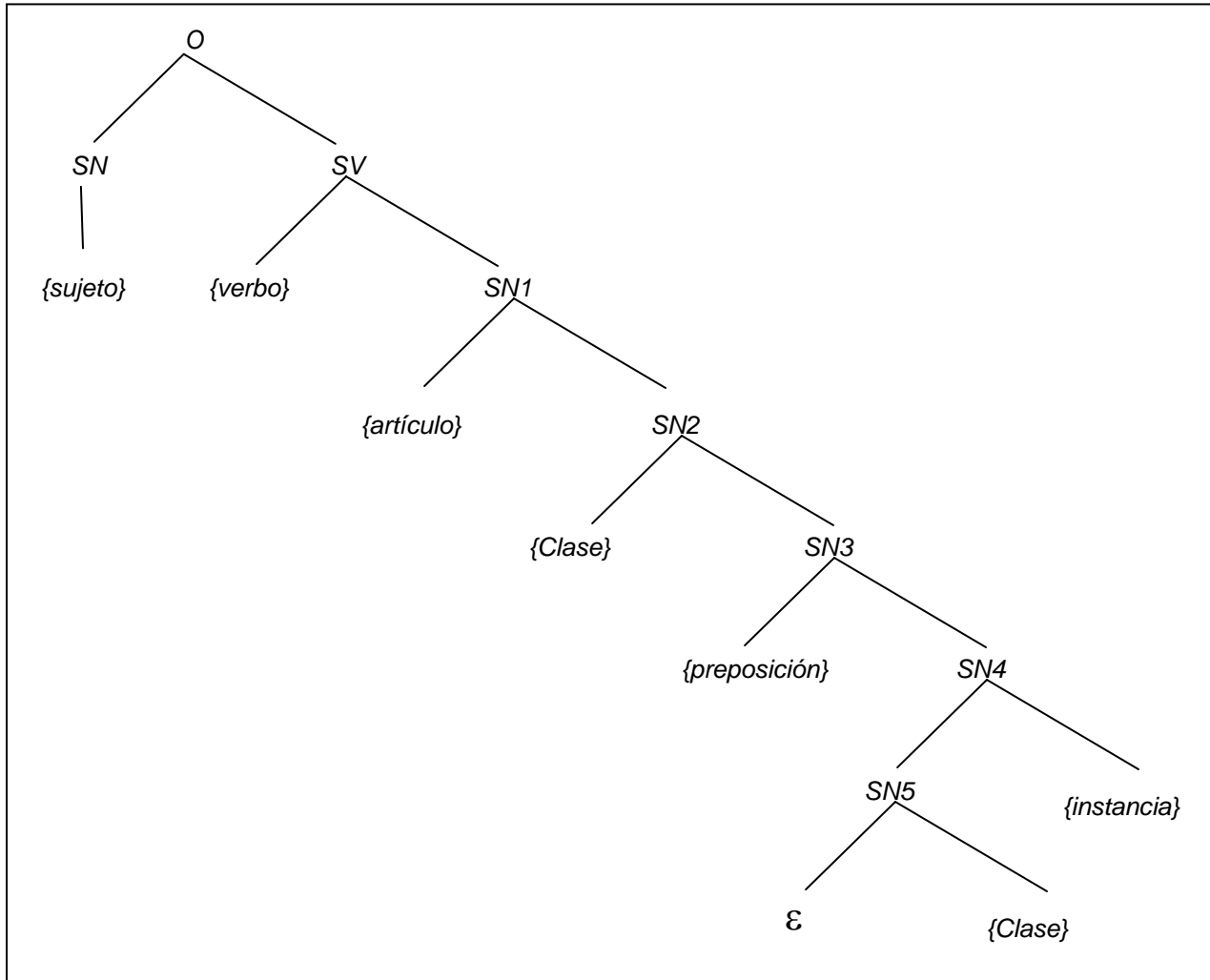


Figura VI.7.- Estructura jerárquica del analizador sintáctico del APIS.

El conjunto de reglas de producción soportado el analizador semántico del SE-APIS es:

$O \rightarrow SN SV$   
 $SN \rightarrow \{sujeto\}$   
 $SV \rightarrow \{verbo\} SN1$   
 $SN1 \rightarrow \{artículo\} SN2$   
 $SN2 \rightarrow \{clase\} SN3$   
 $SN3 \rightarrow \{preposición\} SN4$   
 $SN4 \rightarrow SN5 Instancia$   
 $SN5 \rightarrow \varepsilon \mid \{Clase\}.$



La estructura jerárquica representativa de las reglas de producción del analizador sintáctico se muestra en la Figura VI.7.

El analizador sintáctico del APIS detecta los componentes sintácticos como primer opción en el orden que aparecen en el árbol jerárquico, y adicionalmente el orden de la oración puede cambiar a partir del sintagma nominal SN2, esto gracias a la arquitectura de pizarrón de la máquina de inferencia de *CLIPS*, el cual se activa de una manera *oportunistica* cuando los hechos cumplen con las condiciones de las reglas de inferencia.

El procesamiento de las reglas de inferencia que participan en el análisis sintáctico en el APIS, inician con el análisis de los hechos de asignación de las categorías verbo, sujeto y complemento, continúan con el análisis detallado del complemento y finalizan al iniciar el análisis de la formación de las dependencias conceptuales, como lo muestra la Figura VI.8

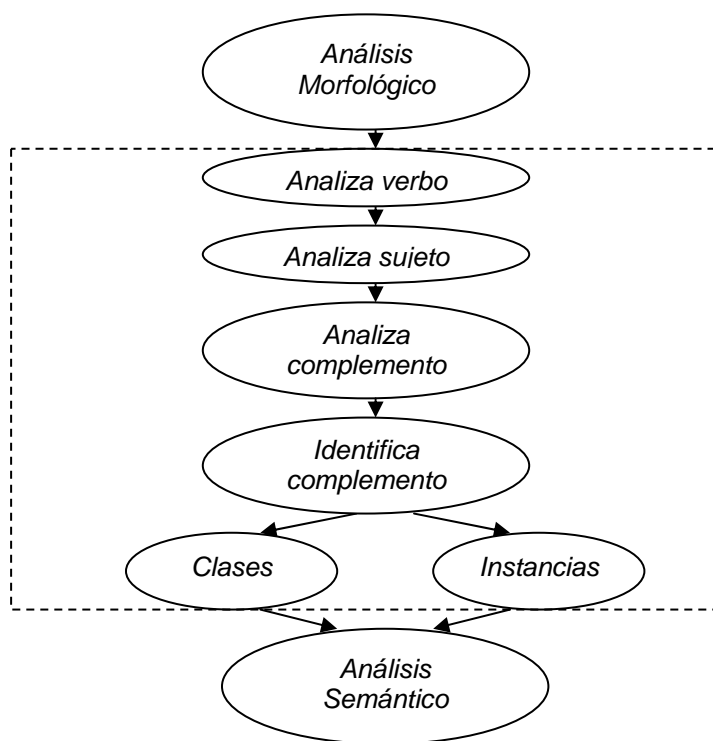


Figura VI.8. Procesamiento del análisis sintáctico del SE-APIS

El modelo del procesamiento sintáctico mostrado en la Figura VI.8 procesa el modelo de *Marcos* de Conocimiento de la Figura VI.4, identificando las clases de conocimiento: Procedimientos, Metodologías, Fases, Actividades, Proyectos así como las instancias de estas mismas.

### **Análisis semántico conceptual del APIS**

El *análisis conceptual* al igual que las gramáticas semánticas es una estrategia diseñada para encontrar en un solo paso tanto la estructura como el significado de una oración en el lenguaje destino. El análisis conceptual lo dirige un diccionario que describe el significado de las palabras como estructuras de dependencias conceptuales **CD**.

El proceso de análisis está fuertemente dirigido por un conjunto de expectativas que se establece sobre la base del *verbo* de la oración. El verbo en las CD es de bajo nivel por lo que se proporciona un mayor grado de potencia predictiva con sus primitivas creadas.

El primer paso en el APIS para realizar una correspondencia entre una oración en Lenguaje Natural y su representación en CD es tomar el verbo y el sujeto extraído con el analizador sintáctico. Se utiliza un diccionario con un conjunto de verbos (ACT) [SHA75], que contiene una entrada por cada verbo que pueda aparecer.

Los verbos contenidos en el PROTOTIPO APIS son: {consulta, reporta, crea, programa}, aunque podría ampliarse el análisis para contener los verbos {controla, evalúa, cierra}.

Una vez elegida la entrada correcta del verbo, el procesador conceptual del APIS analiza el resto de la oración obtenida por el analizador sintáctico buscando los componentes que rellenarán las ranuras vacías de la estructura de la CD. Las estructuras de relleno y ranura fuerte **CD** del APIS se definen de la forma siguiente:

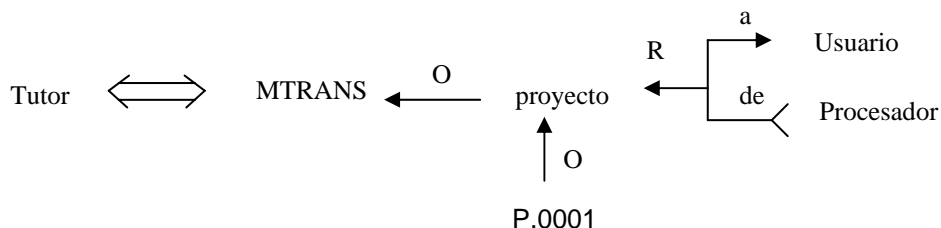
**consulta** [(actor nil) (clase nil) (instancia nil)]  
**reporta** [actor nil) (clase nil) (instancia nil)]  
**crea** [(actor nil) (clase\_from nil) (clase\_to nil) (instancia nil)]  
**programa** [(actor nil)(clase nil) (instancia nil)]

Dado que las dependencias conceptuales definidas en la bibliografía son para sujetos y objetos en movimiento a excepción de MTRANS y MBUILD, entonces las CD del APIS se definieron tomando en cuenta su significado conceptual:

MTRANS → Transferencia de información mental.  
 MBUILD → Construcción de nueva información a partir de la anterior

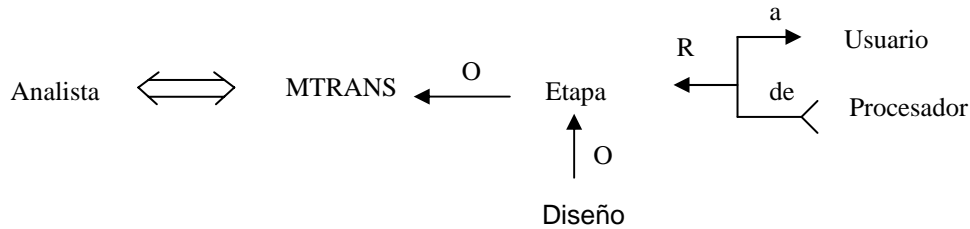
En la dependencia conceptual MTRANS el modelo del sistema de procesamiento humano de la información, se sustituye por un modelo de procesamiento computacional, de tal forma que una oración de consulta de información puede representarse por medio de esta dependencia de la siguiente forma:

*Tutor consulta el proyecto P.00001.*

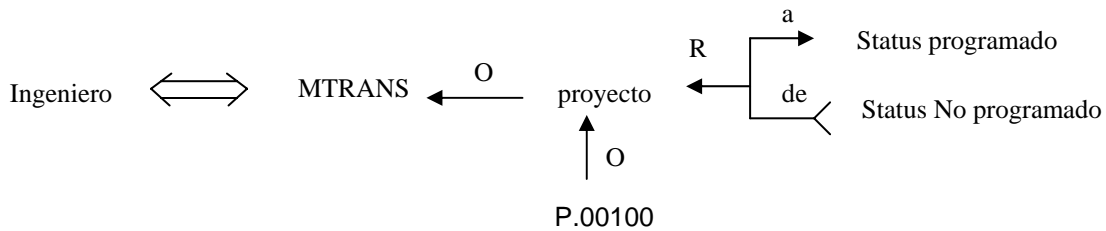


De forma similar se puede utilizar la dependencia conceptual MTRAS para el verbo *reporta* y *programa*, como lo demuestra la siguiente interpretación:

*Analista reporta la etapa de diseño.*

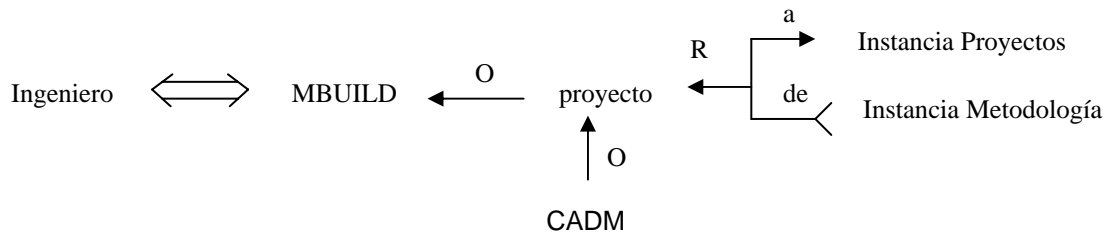


*Ingeniero programa el proyecto P.00100.*



La conceptualización de la dependencia *crea* del APIS se define a partir de la **CD** MBUILD, establecida por Schank en 1975 [SHA75], como lo muestra la siguiente expresión:

*Ingeniero crea un proyecto de metodología CADM*



La definición de las dependencias conceptuales del APIS *consulta()*, *reporta()*, *crea()* y *programa()* se declaran con su verbo natural en español en *CLIPS*, en lugar de las **CD** MTRANS y MBUILD propuestas por Schank.

Para la obtención de la información de la base de conocimientos de Ingeniería de *Software*, se incluyen las dependencias conceptuales en las reglas de inferencia de consulta y reporte, las cuales se aparean con las estructuras condicionales de los *marcos* y hechos de control. Para la activación de las reglas de inferencia el sistema busca instancias de la base de conocimientos que cumplan con las condiciones definidas, si éstas se satisfacen significa que la información en la base de conocimientos se ha encontrado y se obtienen los hechos para enviarse al usuario de acuerdo con los formatos de salida previamente programados.

### VI.3.2. Proceso de Proyectos de Ingeniería de Software del SE-APIS

El ciclo de vida de un proyecto de *software* de acuerdo con las metodologías presentadas en este trabajo se resume en las siguientes fases: *planeación, análisis, diseño, construcción, pruebas y mantenimiento*, así mismo los procedimientos primordiales de la administración del proyecto que son tomadas en cuenta en este trabajo para el desarrollo de los procesos del sistema experto APIS son: *planeación y programación*.

La estructura de *marcos* de la base de conocimientos para soportar las fases y los procedimientos de *planeación y programación* están diseñados en el APIS como lo muestra la Figura VI.4. Adicionalmente, en este apartado se describen las reglas de inferencia para la planeación automática de proyectos de ingeniería de *software* y la programación de las fases y actividades de los mismos.

La **planeación inicial automática de proyectos** de ingeniería de *software* en el APIS se lleva a cabo a partir de la construcción de la dependencia conceptual **crea()** por el procesador semántico. La estructura de ranura y relleno fuerte de la CD *crea* se representa como una estructura condicional de las reglas de inferencia en el motor de inferencia *CLIPS* de la siguiente forma:

**(crea (actor .?actor)(clase\_from ?clase)(clase\_to ?clase)(instancia ?instancia))**

Donde: *actor* es el sujeto que ejecuta la tarea.  
*Clase\_from* es la superclase o marco de las metodologías con la que se construirá el proyecto.  
*Clase\_to* es la clase proyecto que se construirá automáticamente con una versión de plan inicial.  
*Instancia* es el tipo o instancia de metodología y fases con las que se construirá el proyecto.

La incorporación en la parte condicional de la regla de inferencia de la CD *crea()* en conjunto con la estructura de clase "*Metodologías*" dispara la parte consecuente para adquirir los hechos globales que generan de manera automática el proyecto, Figura VI.9.

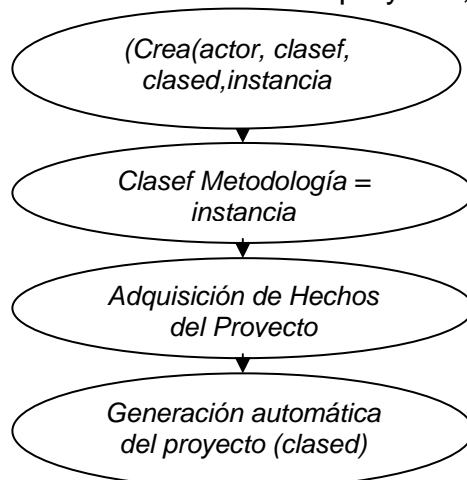


Figura VI.9 Procesamiento de la generación de proyectos, CD *crea()*

De la misma manera, las instancias de la clase WBS del proyecto se generan a partir de las instancias de la clase FASES de la metodología. Las FASES en teoría deben heredar sus atributos a los WBS al tener una relación (**is-a**), sin embargo, esta herencia en el APIS se lleva a cabo mediante un proceso de reglas, debido a que el motor de inferencia *CLIPS* no permite la generación de *marcos* y ranuras(*slots*), de manera dinámica. Por lo que es necesario declarar las reglas de la generación de las WBS para crear cada una de las FASES del proyecto.

De la misma forma se crean las OPERACIONES de cada WBS del proyecto a partir de las ACTIVIDADES de las FASES de la metodología, las cuales heredan sus atributos a partir de un proceso que incluye varias reglas de inferencia. El proceso general para crear de manera automática la planeación de WBS y OPERACIONES de un proyecto de Ingeniería de *Software* se muestra a continuación en la Figura VI.10.

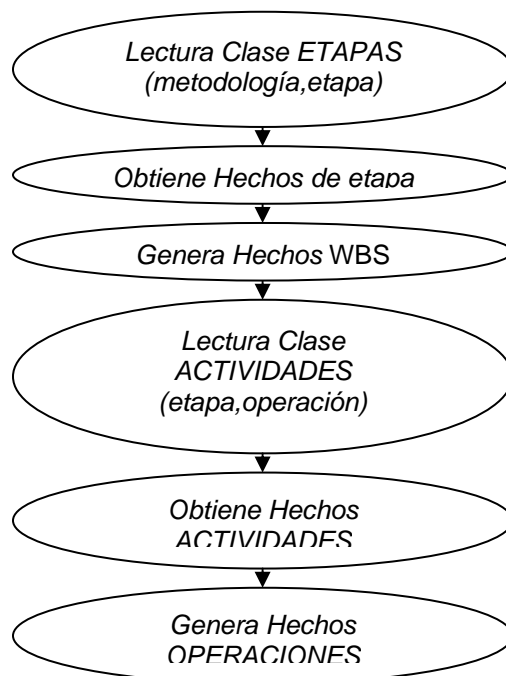
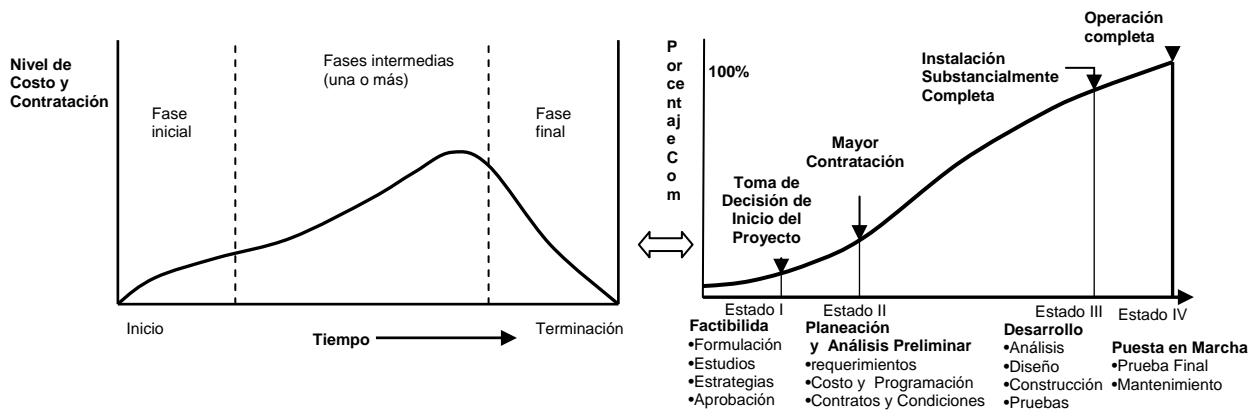


Figura VI.10.- Generación de las WBS y OPERACIONES del Proyecto

Dentro de la creación automática del Proyecto, WBS's y OPERACIONES un campo muy importante es el peso porcentual de las ETAPAS y ACTIVIDADES de la metodología, a partir de éste y de los datos globales proporcionados por el usuario sobre el proyecto se pueden calcular las horas, costo y duración para cada una de las WBS's y OPERACIONES del proyecto de Ingeniería de *Software*.

La forma de determinar estos pesos porcentuales es partiendo de las curvas paramétricas típicas de comportamiento de los proyectos mostradas en la Figura VI.11.

Figura VI.11.- Curvas paramétricas de un proyecto.



Una muestra de datos estadísticos de diferentes proyectos de sistemas de información que cumplen con las curvas paramétricas se presentan en el siguiente cuadro, Figura VI.12.

PROYECTO	CONCEPTO	ANÁLISIS	DISEÑO	CONSTRUC.	PRUEBAS	TOTAL
P.00001	ESFUERZO PERS/MES	14.5	61.0	26.5	16.0	118.0
	COEF. \$	5200	4800	4250	4500	
	COSTO	75,400	292,800	112,625	72,000	552,825
P.00002	ESFUERZO PERS/MES	21.75	91.50	39.75	25.25	178.25
	COEF. \$	5200	4800	4250	4500	
	COSTO	113,100	439,200	168,937	113,625	834,862
P.0003	ESFUERZO PERS/MES	33.50	173.00	69.5	47.17	323.17
	COEF. \$	5200	4800	4250	4500	
	COSTO	174,200	830,400	295,375	212,265	1,512,240
PROMEDIO DE FASE	ESFUERZO PERS/MES	23.25	108.50	45.25	29.47	206.47
	COSTO	120,900	520,200	192,312.33	133,6230	967,642
FACTORES % PESO	ESFUERZO PERS/MES	.11	.53	.22	.14	
	COSTO	.12	.54	.20	.14	

Figura VI.12.- Datos estadísticos de diferentes proyectos

El promedio de los valores de esfuerzo por mes y costo se calculan con la media aritmética de los datos estadísticos de los diferentes proyectos, así mismo los factores de peso se calculan con el peso porcentual de cada uno con respecto al total de las diferentes etapas.

$$\text{Valor promedio de fase}(j) = \frac{(\sum_{i=1}^n \text{val de fase}(i))}{n}$$

Esfuerzo y Costo

$$\text{Factores \% Peso} = \frac{\text{Valor promedio de fase}(j)}{(\sum_{J=1}^n \text{val promedio de fase}(j))}$$

Los valores de Factores de % peso de las fases permiten obtener una curva estadística del comportamiento de este pequeño grupo de proyectos que se muestra en la Figura VI.13.

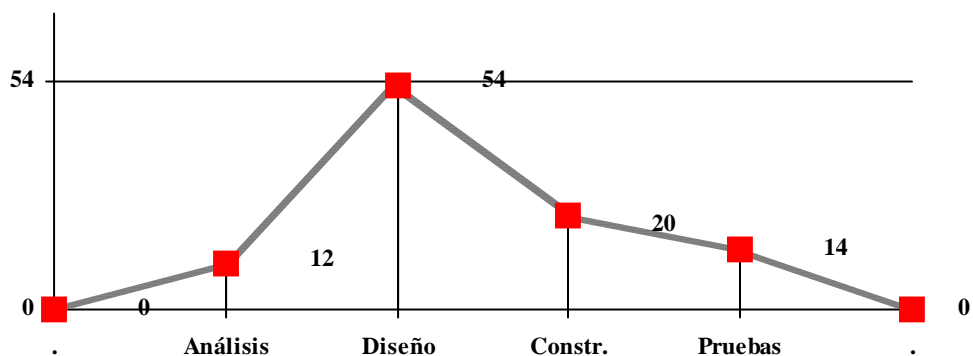


Figura VI.13.- Curva estadística de proyectos de Ingeniería de Software

Al realizar un ajuste de forma heurística a los valores de % de peso de la curva estadística para acercarlas a las curvas paramétricas de los proyectos, se asigna un valor inicial a las instancias de las fases de la metodología CADM de la base de conocimientos del APIS.

CONCEPTO/ PESO	PLANEACIÓN %	ANÁLISIS %	DISEÑO %	CONSTRUCCIÓN %	PRUEBAS %	IMPLEMENTACIÓN %
Esfuerzo	5	20	35	25	10	5
Costo	5	20	35	25	10	5

Figura VI.14.- Pesos porcentuales de las fases de la metodología CADM

Estos % peso se cargan inicialmente en la base de conocimientos, en las diferentes fases de la metodología CADM y se fragmentan en las actividades de cada una de las fases. Estos valores pueden ser actualizados de acuerdo con las diferentes estadísticas de los proyectos de software que se vayan registrando.

### VI.3.3. Aplicación de la Métrica del Modelo de Putnam en los Proyectos

Un ejemplo de la aplicación de Métricas en el SE-APIS se ha desarrollado considerando el Modelo de PUTNAM que nos permite dentro del módulo de planeación automática inicial de los proyectos de Ingeniería de *software* (CD crea()), aplicar las ecuaciones de la curva paramétrica de Rayleigh-Norden, cuyas bases teóricas se presentan en el Capítulo III.

$$L = Ck K^{1/3} td^{4/3} \quad (1) \quad [PUT78]$$

$$K = L^3 / (Ck^3 td^4) \quad (2)$$

$$K = [L \times B^{0.333} / Ck]^3 \times (1/td)^4 \quad (3) \quad [PUT92]$$

La primera ecuación se usa para obtener las líneas de código estimadas en la planeación inicial del proyecto de Ingeniería de *Software*. En esta ecuación el esfuerzo  $K$  puede quedar expresada en función de horas y considerando las horas laborables en un mes, se expresa de la siguiente forma:

$$L = Ck (\text{horas totales del proyecto/horas laborables} * td)^{1/3} td^{4/3} \quad (4)$$

En la base de conocimientos del APIS se consideran 8 horas laborables por día y un valor para  $Ck$  de 8.000 para un buen entorno de desarrollo de *software*, considerando una buena metodología, adecuada documentación y un modo de desarrollo interactivo, así mismo el valor de  $td$  es la duración total en meses del proyecto.

La ecuación (2) del método de Putnam se utiliza durante la ejecución de las fases de diseño, construcción y pruebas para ajustar los valores de esfuerzo y productividad en horas, y se aplica en el momento en que al menos haya construido un prototipo del sistema y se tengan datos cercanos a la realidad de las líneas de código que se deban de realizar dentro del proyecto de ingeniería de *software*.

El modelo de Putnam para obtener el esfuerzo es posible incluirlo en la dependencia conceptual controlar() (en el alcance de este trabajo no está incluida está CD), esta ecuación (2) asociada con la expresión para estimación de valores representada por la curva paramétrica de *distribución beta* de la Figura III.15, queda expresada de la siguiente forma:

$$K = ((La + 4Lm + Lb)/6)^3 / (Ck^3 td^4) \quad (5)$$

De tal forma que la cantidad de líneas de código  $L$ , depende de varios criterios estadísticos que el ingeniero de *software* estime que puedan generarse en la programación de las funciones y/o objetos del sistema de información.



### VI.3.4. Cálculo de la ruta crítica de los proyectos del SE-APIS

El proceso del cálculo de la ruta crítica del SE-APIS, se realiza con la CD programa() y con los Marcos de conocimiento WBS y OPERACIONES del proyecto, los cuales tienen una relación **is-a** con el Marco REDES, que contiene varias estructuras para la definición de precedencias (**PDM**) de las fases y operaciones para el cálculo de la ruta crítica (**CPM**).

Los hechos de conocimiento que definen las precedencias de fases WBS de un proyecto con metodología CADM dentro del Marco REDES se muestran en la Figura VI.15. En esta red se tiene nodos origen y nodos destino, a los cuales se les asigna un número, cada par de nodos se une con un arco que representa cada fase de la metodología CADM, cuya duración ( $t_i$ ) es la obtenida para las WBS con la CD crea().

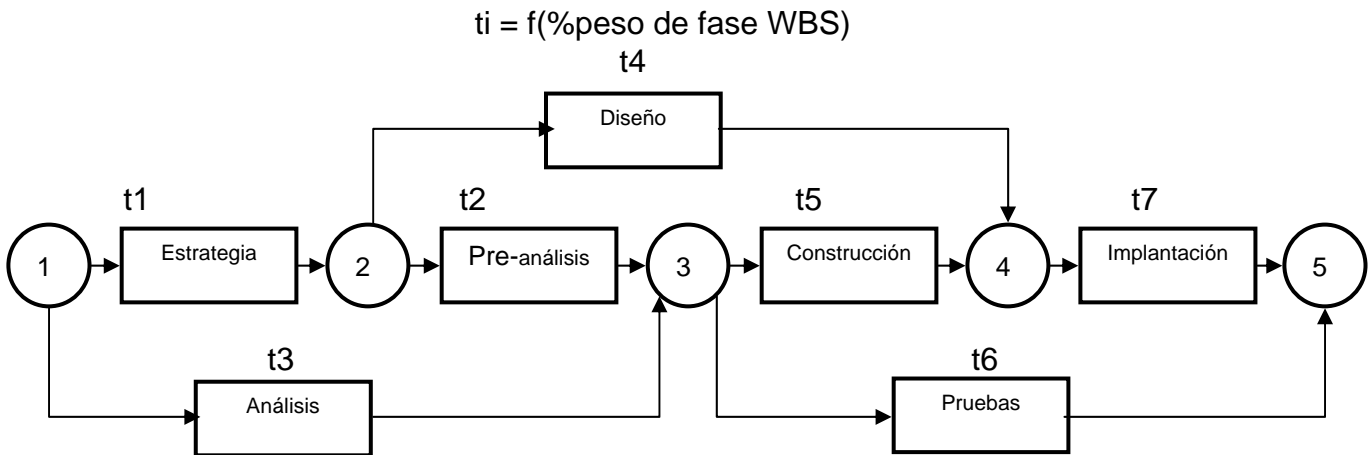


Figura VI.15.- Red de fases WBS de un proyecto de Ingeniería de Software

El cálculo de la ruta crítica de esta red de hechos de conocimiento, es interesante, debido a que los nodos que formaran la ruta crítica cambian de manera dinámica dependiendo de los valores estadísticos de los % peso de las fases WBS.

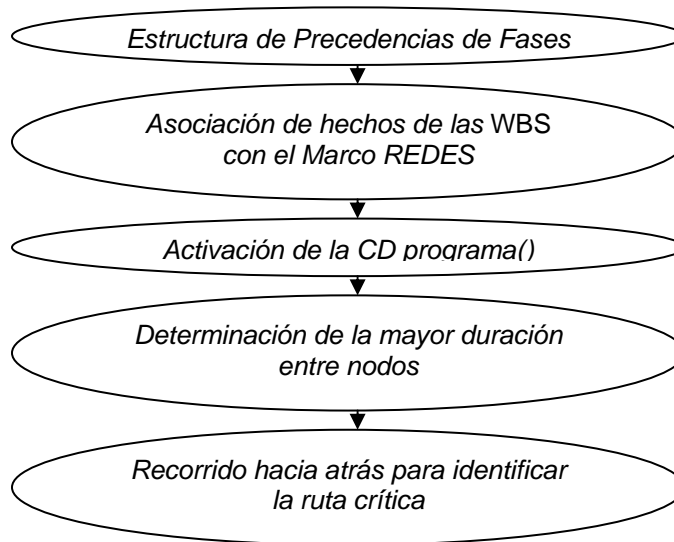


Figura VI.16.- Proceso de cálculo de la ruta crítica en el SE-APIS

La determinación de la ruta crítica se realiza utilizando los hechos de conocimiento del Marco REDES, de acuerdo con el criterio señalado en el capítulo III, considerando la máxima duración acumulada entre nodos:

$$drc(k) = \text{la máxima duración acumulada } (j) + da(k)$$

La identificación de las fases del proyecto que se encuentran en la Ruta Crítica se realiza aprovechando las características del lenguaje *CLIPS*. Iniciando con la identificación en forma recursiva con el método CPM de programación hacia atrás el nodo final de la red, y posteriormente se analizan los nodos iniciales de las actividades que llegan al nodo final, y se identifica el que satisface la duración acumulada mayor. Este nodo se convierte en el nuevo nodo final, el proceso se mantiene mientras haya hechos que disparen al mecanismo de inferencia *oportunistico* del *CLIPS* de acuerdo con las reglas de inferencia programadas, hasta llegar al nodo inicial de la red, momento en el cual se ha logrado obtener la ruta crítica. El diagrama que describe este proceso se muestra en la Figura VI.16.

El desarrollo del cálculo de la ruta crítica en el APIS se realiza en dos partes fundamentales; la primera es la definición de la estructura de *Marcos* y los hechos que describen la red del proyecto a programar. Y la segunda es que está constituida por varias reglas de inferencia que se activan en forma *oportunistica* para obtener la ruta crítica.

#### VI.4. Modelo del proceso del PROTOTIPO SE-APIS

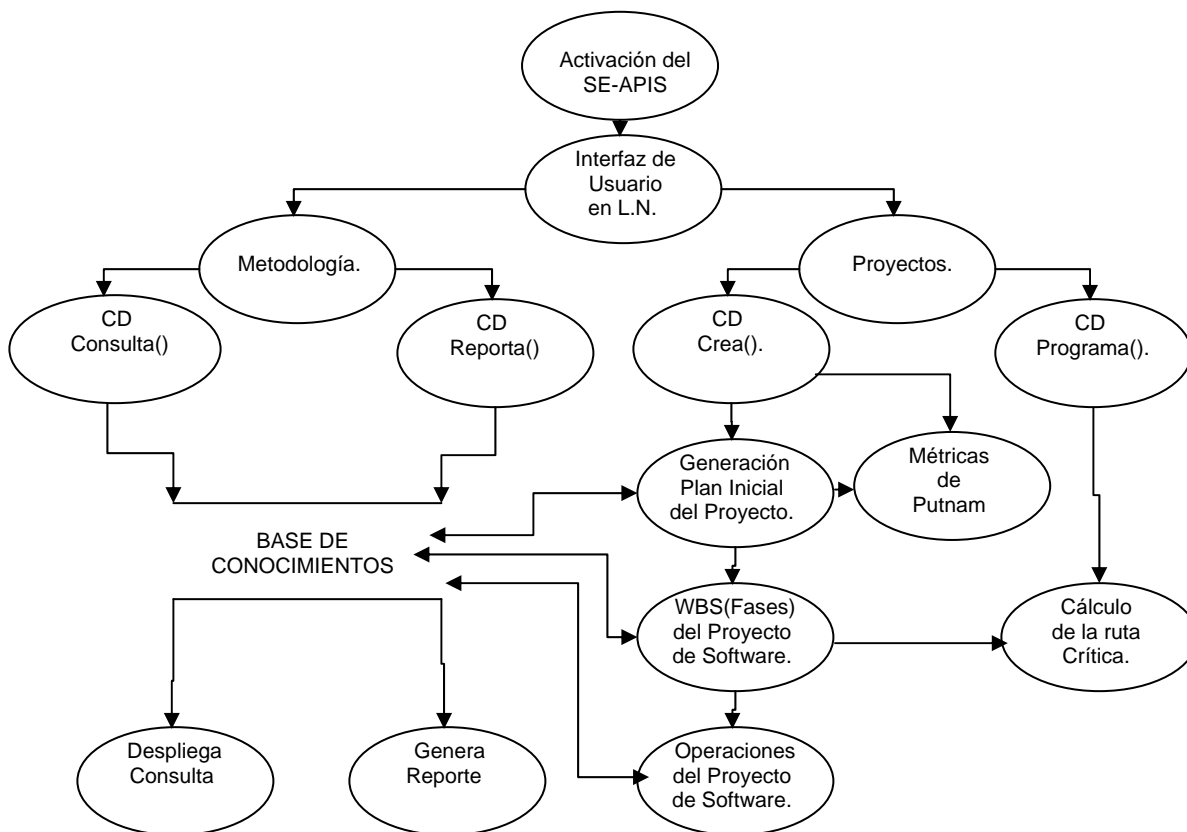


Figura VI.17. Modelo General de Proceso del PROTOTIPO del SE-APIS

El diagrama de la Figura VI.17 muestra de forma general el flujo de procesamiento del Modelo del proceso del PROTOTIPO del Sistema Experto como apoyo a la Administración de Proyectos de Ingeniería *Software* SE-APIS. En este sistema la comunicación con el usuario la forma el procesamiento de Lenguaje Natural con sus tres procesadores: morfológico, sintáctico y semántico. El resultado del análisis morfológico da como resultado la formación de las (CD) dependencias conceptuales consulta() y reporta() para metodologías de desarrollo y las CD crea() y programa() para la Administración de Proyectos, adicionalmente es posible desarrollar otras dependencias conceptuales, tales como controlar() y evaluar(), bajo un Modelo Integral de Herramienta de Ingeniería de *Software* HIS, en el contexto integral descrito en el apartado del alcance del capítulo I.

Las dependencias conceptuales consulta() y reporta() permiten extraer información de la base de datos de conocimiento acerca de las metodologías, etapas y actividades y presentarlas en la pantalla del usuario como resultado de una pregunta en LN. Las CD crea(), permite generar en automático proyectos, fases WBS y operaciones a partir del conocimiento estructurado sobre el modelo de *Marcos* de Conocimiento. De forma simultánea, la CD crea() permite realizar el cálculo de la métrica del modelo de Putnam para obtener las líneas de código a realizar por un programador, a partir de las horas de trabajo planeadas en el proyecto.

Una consideración importante, es que la CD programa() tome los hechos generados en el proyecto sobre la duración de las fases de las WBS, para asociarlas al conocimiento del *marco* REDES y calcular la ruta crítica del proyecto para el CASO de la metodología CADM.

## Capítulo VII. Construcción del PROTOTIPO SE-APIS

### VII.1. Construcción en CLIPS

La construcción del SE-APIS se realizará a partir del modelo diseñado en el capítulo anterior considerando las dos partes fundamentales que lo conforman: el modelo de *Marcos* de Conocimiento y el Modelo de Procesamiento, utilizando para su programación la herramienta para desarrollo de sistemas expertos **CLIPS** (C Language Integrated Production System [CLI97]) de *Software Technology Branch* (STB), de NASA/Lyndon B. Jhonson Space Center, versión 6.05.

Dentro de las características con las que cuenta **CLIPS**, motivo por lo cual fue seleccionado para el desarrollo del SE-APIS, está el facilitar el desarrollo de *software* para modelar el conocimiento o la experiencia humana, permitiendo tres formas de representarlo:

**Reglas de Inferencia**, las cuales son primeramente definidas por medio de conocimiento heurístico basado en experiencia.

**Funciones por definición y funciones genéricas, las cuales son básicamente definidas por conocimiento procedural.**

**Programación Orientada a Objetos** (POO), básicamente definidos por conocimiento procedural. Soportando las cinco características de la POO: definición de clases, message-handlers, abstracción, encapsulamiento, herencia y poliformismo.

Estas características del lenguaje *CLIPS* permiten programar y construir de forma natural las diversas técnicas de la Inteligencia Artificial (capítulo V) con las cuales fue realizado el modelo del SE-APIS: representación del conocimiento con estructuras de relleno débiles *Marcos* (Base de Datos Orientada a Objetos BDOO), estructuras de ranura y relleno fuertes Dependencias Conceptuales CD, procesamiento para gramáticas de Lenguaje Natural LN, Conocimiento heredable, Conocimiento relacional simple y herencias con atributos is-a.

### VII.2. Módulos componentes del SE-APIS

La programación del PROTOTIPO del SE-APIS se realizó en cinco módulos de estructuras y reglas para el procesamiento inferencial, así como reglas de control para la activación de las mismas reglas dentro del mecanismo de inferencia de *CLIPS*. Estos módulos se muestran en la Figura VII.1 y básicamente realizan las funciones siguientes:

- *Métodos2.clp*.- En este módulo está construida la estructura de *Marcos* de Conocimientos (ranuras de relleno débil) de Procesos, Metodologías, Proyectos, Etapas, Fases, Actividades y Operaciones.
- *Instancias.clp*.- Almacena instancias de Hechos de Conocimiento en las estructuras de *Marcos*. La mayor cantidad de Hechos de Conocimiento corresponden a la Metodología

“Método de desarrollo de aplicaciones CASE”(CADM) la cual está descrita con mayor detalle en el anexo B de este trabajo de tesis.

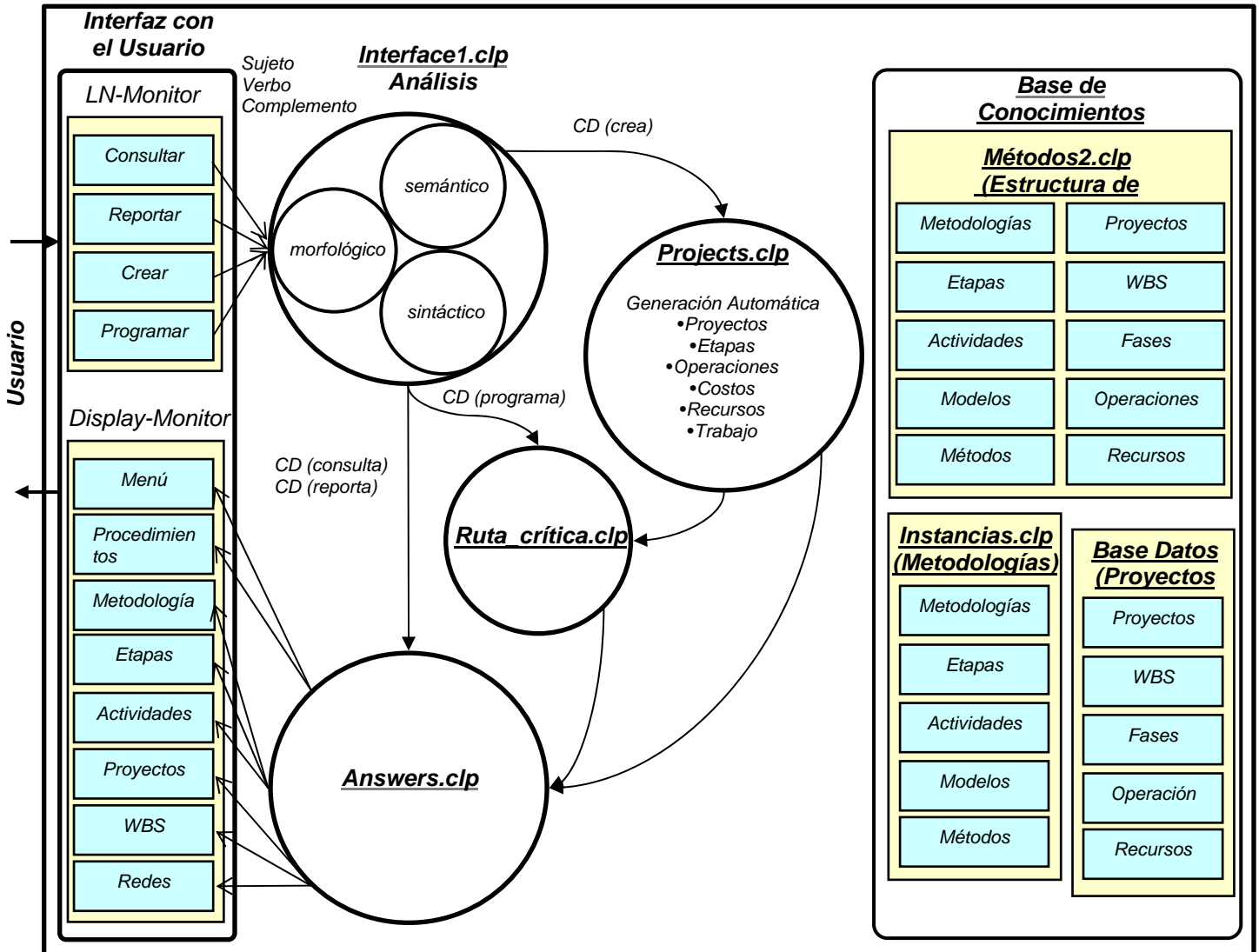


Figura VII.1.- Diagrama de proceso de los módulos de programación del SE-APIS

- *Interface.clp*.- Permite la comunicación con el usuario por medio de la Interfaz con frases en Lenguaje Natural, procesa las oraciones por medio de tres análisis: morfológico, sintáctico y semántico. Como resultado de estos análisis se obtiene la transformación de las frases de Lenguaje Natural limitado en estructuras de Dependencia Conceptual.
- *Projects.clp*.- Genera de forma automática la estructura WBS, Operaciones, Costos, Recursos y Esfuerzos de los Proyectos de Ingeniería de Software a partir de las estructuras de Marcos de Conocimiento y de las Instancias de Hechos, las cuales son tomadas de la Base de Conocimientos como resultado de la activación de la regla de inferencia que contiene la CD crea().

- *Ruta crítica.clp*.- Calcula la ruta crítica de las actividades de los proyectos generados para obtener una programación en tiempo. Las actividades de los proyectos se encuentran almacenadas en archivos con extensión DAT los cuales posteriormente pudieran ser almacenados en una base de datos relacional. En esta versión del PROTOTIPO los hechos para el cálculo de la ruta crítica de los proyectos tiene que ser almacenada manualmente por él EH en la estructura de *Marcos* de las Operaciones y sus precedencias.
- *Answers.clp*.- Es la interfaz de despliegue de información en el monitor del usuario acerca del proyecto y emite información de la Base de Conocimientos directamente cuando se activan las reglas de inferencia de las CD crea() y reporta(), asimismo despliega la información que es reportada por la CD crea() y programa().

**El código completo de la programación del SE-APIS, está documentado en el anexo A, como “código de programación del SE-APIS”, en los siguientes apartados de éste capítulo utilizaremos parte del código en CLIPS para mostrar como se construyó cada uno de los módulos.**

### **VII.3. Construcción del Módulo de la Base de Conocimientos de *Marcos, Métodos2.clp***

La construcción del PROTOTIPO del SE-APIS inicia con este módulo, a partir de la definición de las estructuras de ranura y las instancias de conocimiento del menú de inicio, que en combinación con el módulo de interfaz con el usuario (*Answer.clp*) permite mostrar la primer pantalla del sistema. Las primeras líneas de código sobre el menú inicial son las siguientes:

SISTEMA EXPERTO PARA INGENIERIA DE SOFTWARE  
APIS - BASADO EN CONOCIMIENTOS

Los *deftemplates* y *deffacts* se usan para el encabezado y menú inicial de información del sistema experto para procedimientos de metodologías y administración de proyectos de Ingeniería de *Software*.

```
(deftemplate El-procedimiento-del-Sistema
  (slot es-una))
(deftemplate La-Metodología-es
  (slot orientada-a))
(deftemplate Las-Etapas-a-usar
  (multislot son-las-de))
(deffacts seleccionar
  (El-procedimiento-del-Sistema(es-una "ADMINISTRACIÓN DE PROYECTOS"))
  (El-procedimiento-del-Sistema(es-una "METODOLOGÍAS DE SISTEMAS  ")))
(deffacts metodologías
  (La-Metodología-es(orientada-a nil))
  (La-Metodología-es(orientada-a " OBJETOS  "))
  (La-Metodología-es(orientada-a " CADM  "))
  (La-Metodología-es(orientada-a " PROTOTIPOS")))
```

La definición de la estructura de ranura y relleno fuerte para la dependencia conceptual consulta(), que sirve como lenguaje destino después del análisis semántico del procesamiento del Lenguaje Natural tiene la forma siguiente:

```
(deftemplate consulta
  (field actor (type SYMBOL)(default nil))
  (field clase (type SYMBOL)(default nil) )
  (multifield instancia (type SYMBOL)(default nil)))
```

La representación de las CD reporta(), crea() y programa() tienen estructuras similares.

A continuación se muestra una parte de la definición en CLIPS de la estructura de conocimiento de ranura y relleno débil de la Figura VI.1, aplicando un modelo de relaciones entre clases de objetos o *marcos* de metodología de Ingeniería de *Software*. Iniciando desde un nivel 0 metaclass de CONOCIMIENTOS, nivel 1 metaclass de procedimientos y así sucesivamente hasta llegar al nivel 6 correspondiente a la clase de métodos aplicados a un modelo de actividades de las etapas o fases de una metodología de desarrollo de sistemas.

```
(defclass CONOCIMIENTOS (is-a USER)      ; Nivel 0
  (role concrete)
  (pattern-match reactive)
  (slot sclase (create-accessor read-write)(default nil)))
(defclass PROCEDIMIENTOS (is-a USER)    ; Nivel 1
  (role concrete)
  (pattern-match reactive)
  (slot procedimiento (create-accessor read-write)(default nil))
  (slot objetivo (create-accessor read-write)(default nil)))
(defclass METODOLOGÍAS (is-a PROCEDIMIENTOS) ; Nivel 2
  (role concrete)
  (pattern-match reactive)
  (slot metodología (create-accessor read-write)(default nil))
  (slot id_metodología (create-accessor read-write)(default nil))
  (slot alcance (create-accessor read-write)(default nil)))
(defclass ETAPAS (is-a METODOLOGIAS) ; Nivel 3
  .....
(defclass ACTIVIDADES (is-a ETAPAS)      ; Nivel 4
  .....
(defclass MODELOS (is-a ACTIVIDADES)    ; Nivel 5
  .....
(defclass METODOS (is-a MODELOS)        ; Nivel 6
  .....
```

Una parte de la estructura de conocimientos de ranura y relleno débiles de la Figura VI.3 se muestra a continuación, aplicando un modelo de relaciones entre clases de objetos o *marcos* de administración de proyectos, iniciando desde un nivel 2 clase-PROYECTOS, nivel 3 clase-WBS (*Work BreakDown Structure*), nivel 4 clase-OPERACIONES hasta el nivel 5 clase-

RECURSOS de operaciones. En los *marcos* WBS, operaciones y recursos se ha reemplazado la relación *is-a* por *marcos* formados por tablas relacionales con el fin de facilitar la generación automática de los proyectos con N niveles de WBS.

```
(defclass PROYECTOS (is-a PROCEDIMIENTOS) ; Nivel 2
  (role concrete)
  (pattern-match reactive)
  (slot proyecto (create-accessor read-write)(default nil))
  (slot descripción_proy (create-accessor read-write)(default nil))
  (slot objetivo (create-accessor read-write)(default nil))
  (slot alcance (create-accessor read-write)(default nil))
  (multislot fecha_inicio (create-accessor read-write)(default nil))
  (multislot fecha_final (create-accessor read-write)(default nil))
  (slot dur_proy (create-accessor read-write)(default 0))
  (slot trabajo (create-accessor read-write)(default 0))
  (slot costo (create-accessor read-write)(default 0))
  (slot moneda (create-accessor read-write)(default nil))
  (slot putnam (create-accessor read-write)(default 0))
  (slot lcp (create-accessor read-write)(default 0))
  (slot resp_proy (create-accessor read-write)(default nil)))
(deftemplate WBSS ; (is-a PROYECTOS) ; Nivel 3
  (slot proyecto (default nil))
  (multislot wbs (default nil))b
  (slot id_wbs (default nil))
  (slot fecha_inicio_wbs (default nil))
  (slot fecha_final_wbs (default nil))
  (slot peso_wbs (default 0))
  (slot trabajo_wbs (default 0))
  (slot peso_costo_wbs (default 0))
  (slot costo_wbs (default 0))
  (slot moneda_wbs (default nil))
  (slot dur_wbs (default 0))
  (slot resp_etapa_wbs (default nil)))
(deftemplate OPERACIONES ; (is-a WBSS) ; Nivel 4
  .....
(deftemplate RECURSOS ; (is-a OPERACIONES) ; Nivel 5
  .....
```

#### VII.4. Representación de Instancias y Hechos de Conocimiento, *Instancias.clp*

La representación de Instancias y Hechos utilizadas para probar la base de conocimientos de metodologías de Ingeniería de *Software* y de Administración de Proyectos, se realiza utilizando los *Marcos* definidos en la **Construcción del Módulo de la Base de Conocimientos**. Estas instancias pueden ser actualizadas de acuerdo con nuevas tecnologías y del nivel de conocimiento que se desee que el sistema experto contenga.



Las Instancias del dominio del conocimiento del Sistema Experto como apoyo a la Administración de Proyectos, SE-APIS, se definen en CLIPS de la forma siguiente:

```
(definstances INSTANCIAS_DE_CONOCIMIENTOS
  (clase1 of CONOCIMIENTOS (sclase procedimiento))
  (clase2 of CONOCIMIENTOS (sclase metodologia))
  (clase3 of CONOCIMIENTOS (sclase etapa))
  (clase4 of CONOCIMIENTOS (sclase proyecto)))
```

De la misma forma se definen las dos instancias de procedimientos: metodologías y administración de proyectos:

```
(definstances INSTANCIAS_DE_PROCEDIMIENTOS
  (metodologia1 of PROCEDIMIENTOS (procedimiento metodologías)
  (objetivo "Planear, Administrar y llevar a cabo una implantación optima del desarrollo.
  * de un Sistema de Información cubriendo el total de requerimientos del usuario
  solicitante."))
```

```
(administracion1 of PROCEDIMIENTOS (procedimiento administración)
  (objetivo "Permitir la Planeación, Programación, Presupuestación, Ejecución y Control.
  * de los proyectos para lograr la satisfacción del cliente y cumplir con los objetivos.
  * estratégicos de la empresa."))
```

A continuación se definen las instancias de las metodologías que el SE-APIS contendrá en su base de conocimientos. En este sentido no existe ninguna limitante para incorporar cualquier CASO de metodología que se desee: prototipos, CADM, objetos, clásica, de espiral, etc. Sin embargo para probar el PROTOTIPO solo se han incluido las tres primeras mencionadas, con información muy breve:

```
(definstances INSTANCIAS_DE_METODOLOGÍAS
  (prototipos of METODOLOGÍAS (metodología prototipos)
  (id_metodología 01)
  (alcance "El uso de esta metodología se da cuando normalmente un cliente definirá un
  conjunto de objetivos generales para el software, pero no identifica los requerimientos
  detallados de entrada, procesamiento y salida....."))
```

```
(estructura of METODOLOGÍAS (metodología CADM)
  (id_metodología 02)
  (alcance "Método de desarrollo de aplicaciones CASE."))
```

```
(objetos of METODOLOGÍAS (metodología objetos)
  (id_metodología 03)
  (alcance "Esta orientada a Programación Orientada a Objetos."))
```

Como se menciona en el capítulo IV en su apartado 3, **Biblioteca de CASOS de Metodologías de Desarrollo**, las instancias de etapas y actividades de la metodología que se usará para probar el PROTOTIPO del SE-APIS son las de la metodología CADM, a continuación se describen en CLIPS algunas de ellas:

Instancias de las etapas y actividades de la metodología **CADM**:

El peso de las etapas y actividades para el cálculo automático de los WBS y operaciones en los proyectos se estimaron con los conocimientos de las curvas paramétricas y las estadísticas de varios proyectos de acuerdo con el apartado **VI.3.2. Proceso de Proyectos de Ingeniería de Software del SE-APIS**.

```
(definstances INSTANCIAS_DE_ETAPAS
  (etapa1 of ETAPAS (id_metodología 02)
    (etapa estrategia del negocio)
      (id_etapa 1)
      (descripción_etapa "Estudio de la organización y recursos de la empresa.")
      (peso_etapa 5)
      (peso_costo_etapa 5))
    (act1 of ACTIVIDADES (id_etapa 1)
      (actividad "OBJETIVOS Y ORGANIZACIÓN DEL NEGOCIO ")
      (id_actividad 1)
      (peso_actividad 0.5)
      (peso_costo_actividad 0.5))
    (act2 of ACTIVIDADES (id_etapa 1)
      (actividad "ALCANCE DEL SISTEMA Y MODELO E-R DE ESTRATEGIA")
      (id_actividad 2)
      (peso_actividad 0.5)
      (peso_costo_actividad 0.5))
    .....
    (etapa implementación)
      (id_etapa 8)
    .....
```

Es importante mencionar, que la etapa de mantenimiento está considerada de forma independiente de la curva paramétrica de desarrollo de proyectos de *software* y sus pesos están ponderados de acuerdo con el modelo de PUTNAM para proyectos de ingeniería de *software*.

```
(etapa mantenimiento del sistema)
  (id_etapa 9)
  (descripción_etapa "seguimiento y auditoría de la operación del sistema.")
  (peso_etapa 60)
  (peso_costo_etapa 60))
  (act41 of ACTIVIDADES (id_etapa 9)
    (actividad "CORRECCIÓN DE ERRORES ")
    (id_actividad 1)
    (peso_actividad 10.0)
    (peso_costo_actividad 10.0))
  (act42 of ACTIVIDADES (id_etapa 9)
    (actividad "MODIFICACIÓN DE OBJETOS ")
    .....
```

## VII.5. Construcción de la Interfaz de Procesamiento de Lenguaje Natural, *Interface.clp*

En este módulo se definen los componentes del procesamiento que permiten la interacción con el usuario de acuerdo con el Modelo diseñado en el apartado **VI.3.1. Procesamiento de Lenguaje Natural del SE-APIS**, el cual está compuesto de tres análisis: morfológico, sintáctico y semántico. Esta interfaz soporta oraciones con una gramática limitada, soportando ejemplos con la estructura siguiente:

- > tutor por favor **consulta** el procedimiento de metodología
- > ingeniero **reporta** la etapa de análisis
- > agente **crea** un proyecto de metodología CADM
- > ingeniero **programa** el proyecto P.2000

Derivado del análisis semántico, se obtienen los hechos que han de llenar las ranuras fuertes de las estructuras de las Dependencias Conceptuales (CD): *consulta()*, *reporta()*, *crea()* y *programa()*, las cuales posteriormente son utilizadas en reglas de inferencia que permiten procesar el conocimiento requerido por el usuario.

Los componentes gramaticales que apoyan el procesamiento de las oraciones en Lenguaje Natural están contenidos en varios conjuntos de hechos de conocimiento, definidos en *CLIPS* en la siguiente forma:

```
; sujetos.  
    (sujeto agente) (sujeto tutor)  
    (sujeto analista) (sujeto ingeniero)  
; artículos.  
    (artículo la) (artículo el) (artículo los)  
    (artículo las) (artículo un) (artículo una)  
    (artículo unos) (artículo unas)  
; preposiciones.  
    (prep a) (prep ante)  
    (prep bajo) (prep con)  
    (prep contra) (prep por) (prep de)  
; verbos.  
    (verbo consulta) (verbo reporta)  
    (verbo crea) (verbo programa) (verbo salir)
```

La lectura de las frases del usuario se introducen al sistema mediante un sencillo mecanismo de lectura de información:

```
(defrule receive-data-client  
  ?fact <- (read-data)  
  =>  
  (retract ?fact)  
  (printout t " " crlf)  
  (printout t " dame una orden por favor -> ")  
  (bind ?phrase (readline))  
  (assert (data (explode$ ?phrase))))
```

El Procesamiento del **análisis morfológico** del SE-APIS, se lleva a cabo considerando el modelo definido en la Figura VI.6. iniciando con la separación de palabras de las oraciones que el usuario ha capturado:

```
(defrule separa-palabras
  ?fact1 <- (data ?palabra $?resto)
  ?fact2 <- (cuenta ?valor)
  =>
  (bind ?valor1 (+ ?valor 1))
  (retract ?fact1 ?fact2)
  (assert (palabra (noun ?palabra)(no ?valor1)))
  (assert(cuenta ?valor1))
  (assert (data $?resto))
)
```

El resto del **análisis morfológico** se realiza en paralelo con el **análisis sintáctico**, identificando cada uno de los elementos gramaticales: verbo, sujeto y complemento, además de considerar su posición en la oración con la variable **?valor**. A continuación se muestra el encabezado de las reglas de inferencia que participan en los dos análisis:

```
(defrule detecta-verbo
  ?ptr <- (palabra (noun ?palabra) (no ?valor))
  (verbo ?palabra)
.....
(defrule detecta-sujeto
  ?posv <- (posver ?val)
  ?ptr <- (palabra (noun ?palabra) (no ?valor&:(< ?valor ?val)))
.....
(defrule detecta-complemento
  ?ptr <- (palabra (noun ?palabra) (no ?valor))
.....
```

Para completar el **análisis sintáctico** se realiza un análisis e identificación de los tipos de complemento como lo muestra la Figura VI.8. **Procesamiento del análisis sintáctico del SE-APIS**, estos tipos de complemento son de dos tipos de conocimiento de *clase* y de *instancia*, como lo muestran los encabezados de las reglas de inferencia siguientes:

```
(defrule detecta-clase-conocimiento
  ?c1 <- (complemento ?palabra)
  (object (is-a CONOCIMIENTOS)(sclase ?palabra))
.....
(defrule detecta-instancia-procedimiento
  (complemento ?palabra)
  ?instancia1 <- (object (is-a PROCEDIMIENTOS)(procedimiento ?palabra))
.....
(defrule detecta-instancia-metodología
  (complemento ?palabra)
```

```

?instancia2 <- (object (is-a METODOLOGÍAS)(metodología ?palabra))
.....
(defrule detecta-instancia-etapa
  (not (cuenta 0))
  (complemento ?palabra)
  ?instancia3 <- (object (is-a ETAPAS)(etapa $? ?palabra $?))
.....
(defrule detecta-instancia-proyecto
  (complemento ?palabra)
  ?instancia4 <- (object (is-a PROYECTOS)(proyecto ?palabra))
.....

```

La construcción de la parte del **análisis semántico conceptual del SE-APIS**, se define a partir de los hechos de conocimiento generados a partir de la identificación de los elementos gramaticales y su estructura dentro de los *análisis morfológico y sintáctico*. Los hechos generados, las clases y las instancias se utilizan para activar las reglas de inferencia que construirán la estructura de ranura y relleno fuerte de las dependencias conceptuales consulta(), reporta(), crea() y programa(), como lo demuestra el ejemplo de la construcción de las reglas de inferencia de las CD **consulta()**:

**; Inicia la formación de la CD CONSULTA.**

```

(defrule es-consulta
  ?ref1 <- (referencia consulta)
  =>
  (assert (meta crea-consulta))
  (retract ?ref1)
)
(defrule crea-consulta-limpio
  (meta crea-consulta)
  =>
  (assert (consulta (actor nil)(clase nil)(instancia nil)))
)
(defrule modifica-consulta-actor-clase-instancia
  ?crea-cons <- (meta crea-consulta)
  ?hecho <- (actor ?actor)
  ?clasx <- (clase ?clase)
  ?instx <- (instancia ?instancia)
  ?primitiva <- (consulta (actor nil)(clase nil)(instancia nil))
  =>
  (retract ?crea-cons ?clasx ?instx)
  (modify ?primitiva (actor ?actor)(clase ?clase)(instancia ?instancia))
  (assert (listo on))
)
(defrule checa-si-esta-consulta
  ?listo1 <- (listo on)
  ?primitiva <- (consulta (actor ?actor)(clase ?clas)(instancia ?inst))
  =>
  (retract ?listo1)
)

```

```
(if (neq ?actor nil) then
  (printout t " -> esta listo para responder " ?actor crlf)
  (assert(dependencia-conceptual on))))
```

**; Termina la formación de la CD CONSULTA.**

## VII.6. Generación de Proyectos de Ingeniería de Software y Métrica de Putnam, *Projects.clp*

A partir de la base de conocimientos de metodologías de Ingeniería de Software se generan de manera automática los proyectos estándar de desarrollo de software que a los Líderes de los proyectos les sirve de guía durante el ciclo de vida del desarrollo de los sistemas que se les han encomendado. Esta generación se inicia con la interacción del sistema a partir de la interfaz de Lenguaje Natural, y continúa con la ejecución de la regla de inferencia que contiene la CD **crea()** que ha sido construida en el **analizador semántico conceptual del SE-APIS**.

La activación de la regla de inferencia que contiene la CD **crea()** y que permite generar los proyectos, solicita al usuario datos globales estimados, los cuales son adquiridos por el SE-APIS como nuevos conocimientos para completar la generación y el cálculo de los diferentes componentes del proyecto, la estructura de las fases WBS y sus operaciones, así como de manera paralela una Métrica sencilla para el cálculo de líneas de código utilizando el Modelo de Putnam. La regla de inferencia de la creación de proyectos y los hechos requeridos por el sistema al usuario se muestran a continuación:

*; Creación de la definición del proyecto.*

```
defrule crea-proyectos
  ?cono1 <- (dependencia-conceptual on)
  ?creap <- (crea (actor ?actor)(clase_from ?clas)(clase_to ?clasd)(instancia
?inst))
                (object (is-a METODOLOGÍAS)(metodología ?inst)(id_metodología
?idm))
  =>
  (printout t " Número del proyecto (P.9999) -> ")
  (bind ?proyecto (readline))
  (make-instance projects of PROYECTOS (proyecto ?proyecto))
  (printout t " Descripción del proyecto -> ")
  (bind ?descripción (readline))
  (send [projects] put-descripción_proy ?descripción)
  (printout t " Fecha de inicio (dd/mm/aaaa) -> ")
  (bind ?fi (read))
  (send [projects] put-fecha_inicio ?fi)

  (printout t " Fecha final (dd/mm/aaaa) -> ")
  (bind ?ff (read))
  (send [projects] put-fecha_final ?ff)
  (printout t " Duración en días es -> ")
  (bind ?durd (read))
```

```

(send [projects] put-dur_proy ?durd)
(printout t " Trabajo total en horas    -> ")
(bind ?hhtotal (read))
(send [projects] put-trabajo ?hhtotal)
(printout t " Moneda del costo        -> ")
(bind ?monp (read))
(send [projects] put-moneda ?monp)
(printout t " Costo total del proyecto  -> ")
(bind ?ctotal (read))
(send [projects] put-costo ?ctotal)
(printout t " Responsable del proyecto  -> ")
(bind ?resp (readline))
(send [projects] put-resp_proy ?resp)
(send [projects] print)

```

Así mismo se tienen los hechos adquiridos para realizar el cálculo de las líneas de código a programar para la cantidad de horas de esfuerzo necesarias en la construcción del sistema relacionado con el proyecto generado. Este cálculo se realiza por medio del **Modelo de Putnam**, para diferentes entornos de desarrollo:

```

(printout t " Factor de entorno de desarrollo: " crlf)
(printout t "    1) CK = 2 sin metodología " crlf)
(printout t "    2) CK = 8 con metodología " crlf)
(printout t "    3) CK = 11 con herramientas CASE" crlf)
(printout t "    =>")
(bind ?fact (read))
  (if (eq ?fact 1) then
    (assert(ck 2))
    (bind ?xck 2)
  else
    (if (eq ?fact 2) then
      (assert(ck 8))
      (bind ?xck 8)
    else
      (assert(ck 11))
      (bind ?xck 11)
    )
  )
(printout t " CK = " ?xck crlf)
(send [projects] put-putnam ?xck)

```

### **; Cálculo de líneas de código estimadas**

```

(bind ?l1 (/ ?hhtotal (* 8.0 30.5)))
(bind ?l2 (** ?l1 (/ 1.0 3.0)))
(bind ?l3 (** ?durd (/ 4.0 3.0)))
(bind ?lcp (* (* ?xck ?l2) ?l3))
(printout t crlf " Líneas de código de programación = " ?lcp crlf)
(send [projects] put-lcp ?lcp)

```

Los hechos generados para los datos globales del proyecto se almacenan en un archivo de hechos, que en procesos posteriores permitan su recuperación:

```
(printout t crlf " *** Se salva el archivo del proyecto PROYECTOS.DAT ***" crlf)
(printout t crlf " CUALQUIER TECLA DE LETRA para continuar -> ")
(bind ?ent (read))
(save-instances PROYECTOS.DAT local PROYECTOS)
```

Una vez adquiridos los hechos del proyecto se inicia la generación automática de fases WBSS y de OPERACIONES del proyecto a partir de esta nueva adquisición y de la Base de Conocimientos de Marcos definida en los módulos **Métodos.clp** e **Instancias.clp**, como se muestra en los siguientes fragmentos de código en CLIPS:

**; Genera los WBS del proyecto**

```
(defrule genera-WBS-del-proyecto
  (proyecto on)
  (hereda_proy ?proyecto)
```

.....

```
?numeta <- (num_etapa ?nume)
(object (is-a ETAPAS)(id_metodología ?idm)(etapa $?etapa)(id_etapa ?nume)
  (peso_etapa ?peta)(peso_costo_etapa ?peco))
```

=>

```
(printout t " Se crea la ETAPA -> " ?etapa crlf)
```

.....

```
(printout t " Duración programada en días -> " ?d_wbs crlf)
(assert(WBSS (proyecto ?proyecto)(wbs ?etapa)(id_wbs ?nume)(peso_wbs ?peta)
  (trabajo_wbs ?t_wbs)(peso_costo_wbs ?peco)(costo_wbs ?c_wbs)
  (moneda_wbs ?xmonp)(dur_wbs ?d_wbs)))
```

.....)

**; Existen 9 etapas de metodología estructurada CADM**

```
(defrule obtiene-wbs-proyecto
  (proyecto on)
```

```
(hereda_wbs ?idwbs)
```

```
(WBSS(wbs $?etapa)(id_wbs ?idwbs)(peso_wbs ?peta)(trabajo_wbs
```

?hhwbs)

```
(peso_costo_wbs ?peco)(costo_wbs ?cowbs)(moneda_wbs ?xmonp)
(dur_wbs ?d_wbs))
```

=>

```
(printout t "===== " crlf)
```

```
(printout t " * !!SE GENERAN LAS ACTIVIDADES DE DESARROLLO DE LA !! * " crlf)
```

```
(printout t " * ETAPA " ?etapa crlf)
```

```
(printout t "===== " crlf)
```

.....)

```
(defrule crea-operación-wbss
```

```
(proyecto on)
```

```
(object (is-a ACTIVIDADES)(id_etapa ?idwbs)(actividad ?acti)(id_actividad ?numo)
  (peso_costo_actividad ?pcoac)(peso_actividad ?pact)).....)
```



Los hechos generados para las fases WBSS y las OPERACIONES del proyecto se almacenan en una base de hechos, para posteriormente ser procesados:

```
(save-facts WBSS.DAT local WBSS)
(printout t crlf " *** Se salva el archivo de etapas del SISTEMA WBSS.DAT ***" crlf)
(save-facts OPERS.DAT local OPERACIONES)
(printout t crlf " Se salva el archivo de actividades del SISTEMA OPERS.DAT " crlf)
.....
```

## VII.7. Cálculo de la Ruta Crítica de los Proyectos, *Ruta\_Crítica.clp*

Este proceso se realiza de acuerdo con el modelo diseñado en el apartado VI.3.4. **Cálculo de la ruta crítica de los proyectos del SE-APIS**, considerando la Red de fases WBSS de un proyecto de Ingeniería de *Software* mostrado en la Figura VI.15. una vez que los hechos de duración de las fases del proyecto han sido creados. Esta parte del sistema experto APIS, se activa cuando en el análisis semántico de la Interfaz de Lenguaje Natural termina de llenar la estructura de ranura y relleno fuerte de la dependencia conceptual **programa()**.

La definición de la estructura de los *Marcos* auxiliares para apoyar el cálculo de la ruta crítica, contiene las ranuras (Slot) para la definición de los nodos de la red, sus duraciones, las relaciones de ordenación de 7 fases que componen la red y una marca por cada arco de fase que se encuentra en la ruta crítica.

La estructura siguiente de ranura y relleno débil para el cálculo de la ruta crítica de la red de actividades de un proyecto, forman el *Marco REDES* definido en el modelo de la Figura VI.4, que se describe a continuación:

```
(deftemplate nodo
  (slot número) ;Numeración del nodo
  (slot tiempo) ;Valor de tiempo acumulado mayor asociado al nodo)
(deftemplate actividad
  (slot nombre) ;Nombre de la actividad
  (slot de) ;Nodo origen
  (slot a) ;Nodo destino
  (slot duración) ;Duración de la actividad
  (slot act-cpm) ;Marca si la actividad esta en la ruta crítica)
(deftemplate nodo-final
  (slot nodo-fin) ;Nodo final
  (slot duración-rc) ;Duración total de la ruta crítica)
```

Definición de hechos de las actividades y sus precedencias de la red de 7 fases de WBS para la metodología CADM:

```
(def facts hechos-ruta
  (nodo (número 1)(tiempo 0.0)) ;Nodo inicial tiempo= 0.0
  (nodo (número 2)(tiempo nil)) ;Numera nodos
  (nodo (número 3)(tiempo nil))
  (nodo (número 4)(tiempo nil))
  (nodo (número 5)(tiempo nil))
```

(actividad (nombre estrategia) (de 1) (a 2) (duración 0) (act-cpm no))  
 (actividad (nombre analisis) (de 1) (a 3) (duración 0) (act-cpm no))  
 (actividad (nombre preanalisis) (de 2) (a 3) (duración 0) (act-cpm no))  
 (actividad (nombre diseño) (de 2) (a 4) (duración 0) (act-cpm no))  
 (actividad (nombre pruebas) (de 3) (a 5) (duración 0) (act-cpm no))  
 (actividad (nombre construccion) (de 3) (a 4) (duración 0) (act-cpm no))  
 (actividad (nombre implementacion) (de 4) (a 5) (duración 0) (act-cpm no)))

Activación de las reglas de inferencia para el cálculo de la ruta crítica por medio de la dependencia conceptual **programa()**:

```
(defrule programa-proyectos
  ?cono1 <- (dependencia-conceptual on)
  ?progp <- (programa (actor ?actor)(clase ?clas)(instancia ?inst))
=>
  (assert (progra ?inst))
  (retract ?cono1 ?progp))
```

**Mediante la asociación de hechos de duración de las WBS y de hechos de duración de las actividades definidas en la estructura del Marco REDES, se establece la interfaz con el proyecto generado con la CD crea() y el cálculo de la ruta crítica de la red del proyecto que se activa con la CD programa().**

```
(defrule actualiza-dur-acti
  (declare (salience 250))
  (progra ?inst)
  ?acti <- (actividad (nombre ?act)(duración ?d)(de ?inicio)(a ?fin)(act-cpm ?arc))
  ?wbsr <- (WBS1 ?act ?d_wbs)
=>
  (printout t " WBS: " ?act " ,duración " ?d_wbs crlf)
  (modify ?acti(duración ?d_wbs))
  (retract ?wbsr))
```

Las reglas de inferencia para detreminar el cálculo de la ruta crítica siguen el algoritmo de la mayor duración expresado como: **drc(k) = la máxima duración acumulada (j) + da(k)**, aplicandolo a un algoritmo recursivo que recorre toda la red formado una red reducida con la duración de tres arcos que llegan a los nodos:

```
(defrule etiqueta-nodo ;Identificación de nodos con el mayor
  (declare (salience 200)) ;Tiempo acumulado, los cuales asocian
  (progra ?inst) ;Proyecto a programar con el cálculo de la ruta crítica
  (nodo (número ?inicio) (tiempo ?t&~nil)) ;Define un nodo inicial con tiempo t, no nulo.
```

Los datos de la actividad 'actividad' se comparan con el tiempo t y se asigna a su nodo 'fin' la duración acumulada si es mayor que otra anterior. Se generan todas las combinaciones posibles.

```
(actividad (nombre ?act) (duración ?d) (de ?inicio) (a ?fin))
(not (and
      (nodo (número ?inicio2&?inicio) (tiempo ?t2&~nil))
      (actividad (nombre ?act2&~?act)(duración ?d2)(de ?inicio2)(a ?fin))
      (test (> (+ ?t2 ?d2) (+ ?t ?d))))))
```

```
?nodo <- (nodo (número ?fin) (tiempo ?t3&: (or (eq ?t3 nil)
                                                (< ?t3
                                                 (+ ?t ?d))))))
```

=>

(modify ?nodo (tiempo (+ ?t ?d))) ;Etiqueta nodos con la duración mayor acumulada

Una vez identificado las mayores duraciones en los nodos de la red, se activa una regla de inferencia que identifica al nodo final con la mayor duración acumulada. Un nodo final es aquel que no tiene sucesores.

```
(defrule duración-total ;Identifica el nodo con duración mayor y lo registra
  (declare (salience 100))
  (progra ?inst)
  (nodo (número ?fin) (tiempo ?d&~nil))
  (not (nodo(tiempo ?d1&~nil&: (> ?d1 ?d))))
```

=>

```
(assert(nodo-final (nodo-fin ?fin) (duración-rc ?d)))
(assert (final ?fin) (total ?d)) ;variables para nodo final y su duración acumulada
```

A partir de la identificación del nodo final se realiza un recorrido recursivo de la red hacia atrás hasta llegar al nodo inicial, para identificar cuáles son los nodos y los arcos dentro de la ruta crítica del proyecto y posteriormente se activa una regla para imprimir la ruta obtenida.

```
(defrule actividades-rc
  (declare (salience 50))
  ?nodfin <- (nodo-final(nodo-fin ?nfin)(duración-rc ?drc))
  ?actrc <- (actividad(de ?n1)(a ?n2&?nfin)(duración ?da)
            (act-cpm ?arc&no)(nombre ?nom))
            (nodo(número ?nx&?n1)(tiempo ?d&~nil))
            (test(eq (- ?drc ?da) ?d))
```

=>

```
(modify ?actrc(act-cpm si))
(modify ?nodfin(nodo-fin ?n1)(duración-rc (- ?drc ?da)))
(printout t crlf " act-rc " ?nom " ,duración act " ?da
           " ,nodos: de " ?n1 " a " ?n2 crlf)
(printout t " nuevo nodo final: " ?n1 " ,duración acum. " (- ?drc ?da) crlf)
```

)

```
(defrule impresion-actividades
```

```
.....)
```

## VII.8. Respuestas y presentación de resultados para el usuario, *Answer.clp*

En la interfaz con el usuario también está considerado un módulo de respuestas, las cuales son realizadas al activarse las reglas de inferencia de los menús del sistema experto y de las reglas que contienen las dependencias conceptuales creadas con el análisis semántico conceptual del APIS. A continuación mostramos los encabezados de las reglas de inferencia que despliegan información y resultados en la pantalla del usuario:

### Encabezados del menú de inicio del SE-APIS

```
(defrule imprime-encabezado
  (declare(salience 10))
  =>
  (printout t "===== " crlf)
  (printout t crlf)
  (printout t "===== " crlf)
  (printout t "
                TUTORIAL PARA" crlf)
  (printout t "          " crlf)
  (printout t "          DESARROLLO DE SISTEMAS DE INFORMACIÓN" crlf)
  (printout t "          PROCEDIMIENTOS CONOCIDOS" crlf)
  .....
```

### Consulta de la descripción de procedimientos de metodologías y de administración de proyectos:

```
(defrule consulta-procedimientos
  ?cono1 <- (dependencia-conceptual on)
  ?cons1 <- (consulta (actor ?actor)(clase ?procs)(instancia ?inst))
            (object (is-a PROCEDIMIENTOS)(procedimiento ?inst)(objetivo ?obj))
  =>
  (retract ?cono1 ?cons1)
  (if (neq ?obj nil) then
    (printout t "===== "
  crlf)
  (printout t "*" PROCEDIMIENTO : " ?inst crlf)
  (printout t "*" OBJETIVO      : " crlf)
  (printout t "===== "
  crlf)
  (printout t "*"
                TIPOS DE METODOLOGÍAS" crlf)
  .....
```

### Consulta de la descripción de tipos de metodologías:

```
(defrule consulta-metodologías
  ?cono2 <- (dependencia-conceptual on)
  ?cons2 <- (consulta (actor ?actor)(clase ?mets)(instancia ?inst))
            (object (is-a METODOLOGIAS)(metodología ?inst)(alcance ?alc))
```

```

=>
  (retract ?cono2 ?cons2)
  (if (neq ?alc nil) then
    (printout t "=====
crlf)
  (printout t "* METODOLOGÍA : " ?inst crlf)
  (printout t "* ALCANCE : " crlf)
  ....)

```

**Consulta de las etapas de las metodologías, en el PROTOTIPO se tiene particularmente el CASO de la metodología CADM:**

```

(defrule consulta-etapas
  ?cono3 <- (dependencia-conceptual on)
  ?cons3 <- (consulta (actor ?actor)(clase ?etaps)(instancia ?inst))
  (object (is-a ETAPAS)(etapa $?first ?inst $?last)(descripción_etapa ?dese))
=>
  ....
  (printout t "=====
crlf)
  (printout t "* ETAPA : " (implode$ ?list) crlf)
  ....)

```

**Reporte de las etapas de metodologías, particularmente de la metodología CADM:**

```

(defrule reporta-etapas
  ....
=>
  ....
  (printout t "=====
crlf)
  (printout t "* ETAPA : " (implode$ ?list) crlf)
  (printout t "=====
crlf)
  (printout t "* NO.ACT. ACTIVIDAD % PESO" crlf)
  ....)

```

En este módulo se presentan los reportes básicos del PROTOTIPO del SE-APIS, sin embargo se pueden generar tantos informes de resultados y reportes como se deseen obtener de la base de conocimientos almacenada en estructuras de hechos e instancias de *Marcos*, así como de la base de datos generada para los proyectos.

**El código completo de la programación del SE-APIS, se encuentra en el anexo A de la tesis como "código de programación del SE-APIS", almacenado en un disco compacto, que se entregará como complemento adjunto al trabajo de tesis escrito.**

## VII.9. Métricas, número de reglas de inferencia y líneas de código del SE-APIS

En este apartado se presenta un resumen por módulo de la cantidad de estructuras, reglas de inferencia y líneas de código (LC), que se codificaron en la construcción del SE-APIS. Estos datos aplicados al modelo de Putnam nos permiten calcular el esfuerzo en personas mes o en horas estimadas necesarias para desarrollar los programas en *CLIPS* del PROTOTIPO para el SE-APIS. La tabla de la Figura VII.2 muestra un resumen de estos datos, incluyendo el valor del esfuerzo K, aplicando la ecuación 2 del apartado VI.3.3. “Aplicación de la Métrica del Modelo de Putnam en los Proyectos”.

$$K = L^3 / (Ck^3 td^4) \quad (2)$$

Nombre de Módulo	Descripción de Módulo	Num. Estructuras De Marcos	Num. Reglas de Inferencia	LC	Td días	K , Ck=8 Esfuerzo	Horas K*8*Td
Métodos	Base de Conocimientos de Marcos	23	0	219	20	.13	20.8
Instancias	Instancias y Hechos de Conocimiento	3	0	326	20	.42	67.2
Interfase	Interfaz de Procesamiento de Lenguaje Natural	2	35	320	25	.16	32.0
Projects	Proyectos de Ingeniería de Software y Métrica de Putnam	0	6	188	20	.10	16.0
Ruta_Critica	Cálculo de la Ruta Crítica de los Proyectos	4	7	100	20	.02	3.2
Answer	Respuestas y presentación de resultados	0	10	26	10	.004	.32
Totales		32	58	1179	115		139.52

Figura VII.2.- Métricas del SE-APIS

## VIII. Evaluación de resultados del PROTOTIPO SE-APIS

En este capítulo se realizarán varios casos de prueba que permitan evaluar el comportamiento del SE-APIS de acuerdo con los objetivos y resultados esperados, que se plantearon en el capítulo I de introducción en este trabajo de tesis. Asimismo se evaluarán las estructuras y procesos de los módulos del modelo diseñado para el PROTOTIPO del SE-APIS.

Aún cuando se debe tratar que un modelo esté en su totalidad libre de errores y muestre un comportamiento perfecto, esto es difícil de lograr en la realidad. Los conceptos de verificación y validación no se deben considerar como variables de decisión binaria en el cual los modelos son absolutamente válidos o no válidos, debido a que son solo representaciones o abstracciones de la realidad. Sin embargo es necesario especificar claramente el nivel de desempeño de los sistemas.

### VIII.1. Diseño de pruebas del SE-APIS

Existen dos métodos de evaluación que se pueden aplicar para la verificación y validación de SE-APIS; realización de pruebas cualitativas y pruebas cuantitativas, las que se pueden aplicar para cada uno de los módulos del SE-APIS y en su funcionamiento integral.

#### Evaluación Cuantitativa:

La evaluación cuantitativa se aplicará exclusivamente al módulo de Generación de Proyectos de Ingeniería de *Software* y Métrica de Putnam, *Projects.clp*, mediante la realización de la comparación de pares de cantidades de datos calculados por el SE-APIS, y datos calculados fuera de éste, de acuerdo con la siguiente expresión:

$$\text{Desviación } (i) = / X_i - Y_i / \quad (6)$$

Esta comparación se aplicará para los valores de horas y líneas de código totales de la tabla mostrada en la Figura VII.2, así como a los costos y duración obtenidas a partir de los factores de % peso que se utilizan para cada etapa de la metodología y fases WBS de los proyectos en el SE-APIS para el módulo *Projects.clp* y la dependencia conceptual *crea()*. El formato de la tabla en donde se registrará el resultado de la evaluación cuantitativa se presenta en la Figura VIII.1.

#### Evaluación Cualitativa:

Dentro de éstas pruebas se realizará una evaluación que permita observar en la medida de lo posible el comportamiento y funcionamiento independiente de cada módulo, así como la verificación del buen funcionamiento de las interfaces entre módulos, para validar su comportamiento integral.

Etapa	Datos calculados <b>dentro</b> del SE-APIS					Datos calculados <b>fuera</b> del SE-APIS			
	%Peso	Horas	Costo	Duración	LC	Horas	Costo	Duración	LC
Análisis									
Diseño									
Programación									

Figura VIII.1.- Formato para comparación de valores calculados por el SE-APIS

El formato de la tabla en donde se registrará el resultado de la evaluación cualitativa de cada módulo y de sus interfaces, utilizando diversas dependencias conceptuales (CD) se presenta en la Figura VIII.2.

Módulo	Interfaz	CD	Casos de prueba en LN	Acierto
Métodos2.clp				
Instancias.clp				
Interface.clp				
Projects.clp				
Ruta_crítica.clp				
Answers.clp				
Total de aciertos				

Figura VIII.2.- Formato para pruebas de validación modular e interfaces del SE-APIS

Adicionalmente a las pruebas anteriores, se realizará una validación de defectos de funcionamiento de los diferentes módulos, para identificar los casos de falla que pueden surgir en el procesamiento del conocimiento del PROTOTIPO del SE-APIS. La tabla de la Figura VIII.3, muestra el formato con el que se validaran los defectos por módulo.

Módulo	Descripción del defecto
Total de defectos	

Figura VIII.3.- Formato para verificación de defectos de los módulos del SE-APIS

Por último se listarán las excepciones, con las cuales el PROTOTIPO no cumple, con el fin de dejar documentado que tipo de casos no tendrán respuesta para los requerimientos de información del usuario.

### **Escenarios de prueba para las Evaluaciones Cuantitativa y Cualitativa del SE-APIS**



Es conveniente mencionar que los resultados obtenidos tanto de la evaluación cuantitativa como de la cualitativa se registran de acuerdo al funcionamiento del SE-APIS, en cuatro escenarios o grupos de pruebas:

- 1) Interfaz de Lenguaje Natural
- 2) Consulta y reportes de Procedimientos y Metodologías
- 3) Generación de Proyectos y Métricas de Putnam
- 4) Cálculo de la ruta crítica de Proyectos

## VIII.2. Ejecución del SE-APIS en CLIPS

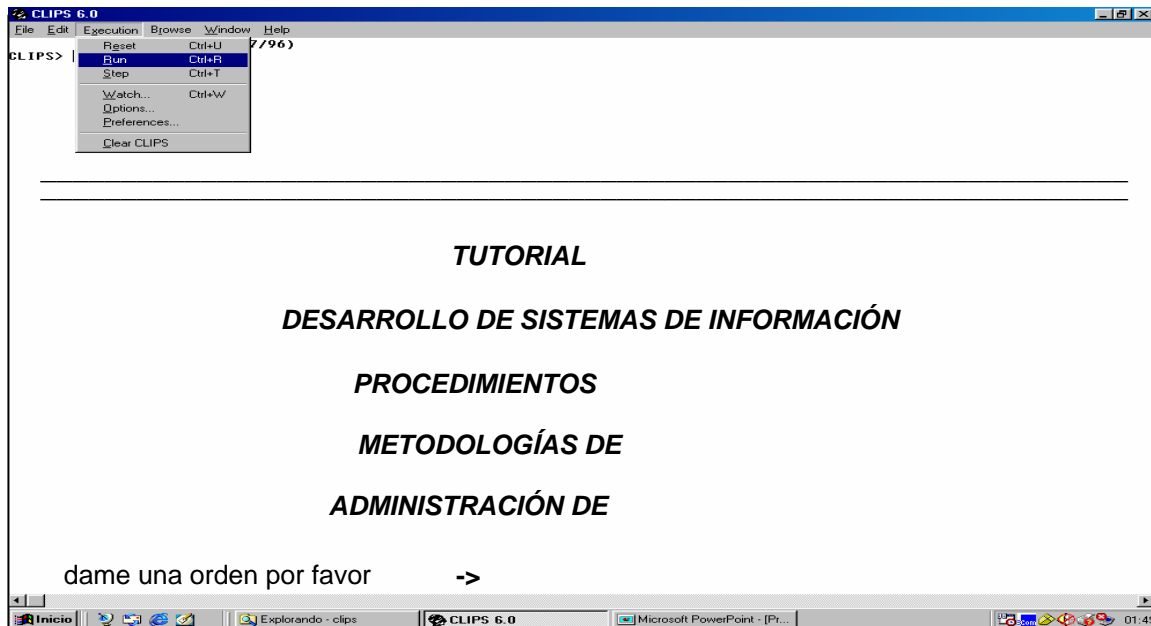
Las corridas de ejecución del SE-APIS tal y como se encuentra actualmente programado se realizan en el menú del *software CLIPS* de acuerdo con los siguientes pasos:

1. Ejecución del programa *CLIPSW32* .
2. Borrado del pizarrón de hechos mediante la selección de la opción *Execution->Clear Clips*.
3. Selección de los parámetros que se desean monitorear durante la ejecución del Sistema Experto *Execution->Watch*. Si se desea únicamente observar la interacción con el sistema se recomienda sólo activar la opción *Compilations*.
4. Carga de los módulos programados con la opción *File->Load Batch*, en el siguiente orden: *Metodos2.clp*, *Instancias.clp*, *Intefase1.clp*, *Projects.clp*, *Ruta\_crítica.clp* y *Answers.clp*.
5. Activar los hechos iniciales del SE-APIS mediante *Ejecución->Reset*.
6. Ejecutar el Sistema Experto con *Ejecución->Run*.

Una vez realizados estos pasos el sistema comenzará con la presentación de un sencillo menú inicial y presentará una *prompt* para ponerse a las órdenes del usuario.

## Inicio del SE-APIS

Cuando se inicia la ejecución del SE-APIS, la primer pantalla que aparece es el MENÚ principal con el contenido que es posible procesar: Metodologías de Sistemas y Administración de Proyectos.



El sistema solicita al usuario que se le indique una orden en lenguaje natural limitado, en este caso se tienen programadas las acciones de consulta, reporte, crear de forma automática proyectos y programar ruta crítica.

En los siguientes apartados realizaremos la aplicación de las Evaluaciones Cuantitativa, Cualitativa y de defectos del SE-APIS, bajo los cuatro escenarios de prueba mencionados con anterioridad, así mismo realizaremos una lista de excepciones de la funcionalidad del SE-APIS por cada escenario de pruebas. El resultado de las pruebas se registrará el final de cada escenario o grupo de pruebas, en los formatos diseñados para este fin.

### VIII.3. Escenario de evaluación de la Interfaz de Lenguaje Natural

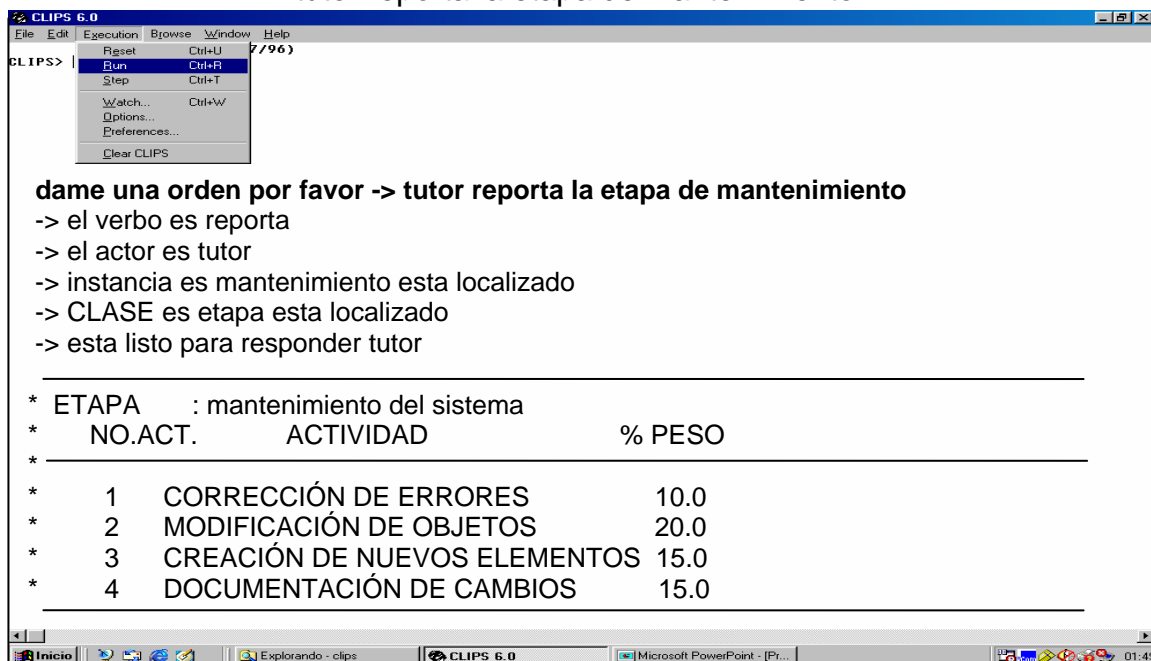
La evaluación de la interfaz de Lenguaje Natural se llevará a cabo realizando pruebas con tres dependencias conceptuales `reporta()`, `crea()` y `programa()` para verificar que el PROTOTIPO del SE-APIS responde correctamente a las ordenes que se le dan mediante oraciones que cumplan con la estructura gramatical definida en el modelo diseñado del apartado VI.3.1.

**SN SV artículo Clase preposición ε Instancia**  
**SN SV artículo Clased preposición Clasef Instancia**

#### Caso 1) Validar la dependencia conceptual CD `reporta()`

En este caso de prueba lo que se requiere es validar que la interfaz de Lenguaje Natural responda correctamente a la gramática de contexto libre de la CD reporta(), identificando que las tres etapas de procesamiento: análisis morfológico, análisis sintáctico y el análisis semántico funcionen adecuadamente para generar la respuesta de conocimiento requerida por el usuario. La frase que se desea que el usuario teclee en lenguaje natural para verificar este caso es:

“tutor reporta la etapa de mantenimiento”



Es claro que en la oración del Caso 1, el nombre de la instancia de la metodología no es necesario que se teclee explícitamente por parte del usuario. Aparentemente en el SE-APIS no existe confusión de que metodología se trata, de CADM o de la metodología de prototipos. Esto es debido que solo tenemos almacenadas instancias de conocimiento sobre las etapas y actividades de la metodología CADM. Bajo una situación en donde se tuvieran las instancias de etapas y actividades en los Niveles 3 y 4 del modelo de Marcos de conocimiento de otras metodologías, se tendría que ampliar la estructura gramatical de las oraciones de lenguaje natural para interpretar oraciones con la estructura:

***SN SV artículo CLASE-m instancia-m instancia-M preposición CLASE-M***

Para interpretar una oración completa como:

“analista reporta la etapa de estrategia de la metodología CADM” ó  
 “analista reporta la etapa de mantenimiento de la metodología de espiral”

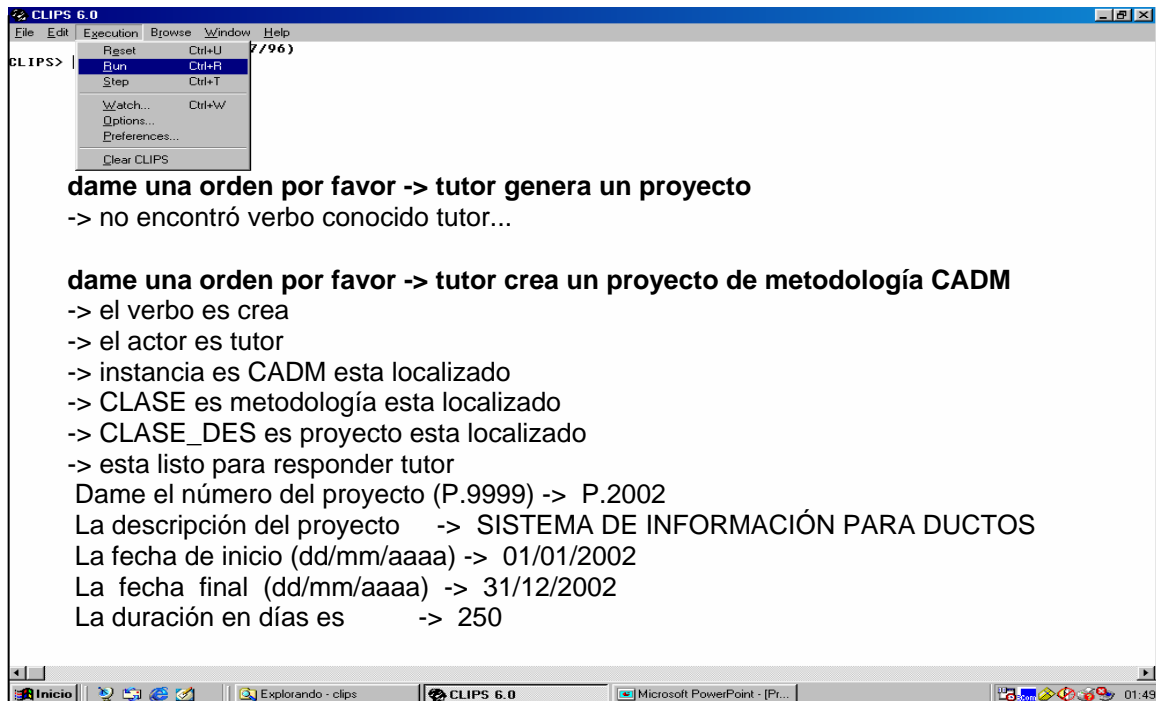
donde: CLASE-m e instancia-m son la clase y la instancia de las etapas y  
 CLASE-M e instancia-M son la clase y la instancia de la metodología.

Esta es una excepción del SE-APIS, que será incluida en la lista de evaluación al termino de este escenario de casos de prueba.

## Caso 2) Validar la dependencia conceptual CD crea()

En este caso se valida que la dependencia conceptual crea() se genera adecuadamente después de los diferentes análisis del procesamiento de Lenguaje Natural, para lo cual se utiliza la oración:

“tutor crea un proyecto de metodología CADM”



En este caso se utiliza una oración con una estructura gramatical:

### SN SV artículo Clased preposición Clasef Instancia

En la cual el SV (Sintagma Verbal) es “crea”, de aquí que el SE-APIS cuando detecta un verbo como “genera”, despliega un mensaje como el mostrado en la pantalla anterior: “no se encontró verbo conocido tutor”. De la misma forma si el SN (sujeto), alguna de las Clases o la Instancia no son conocidas se despliegan los mensajes de advertencia correspondientes.

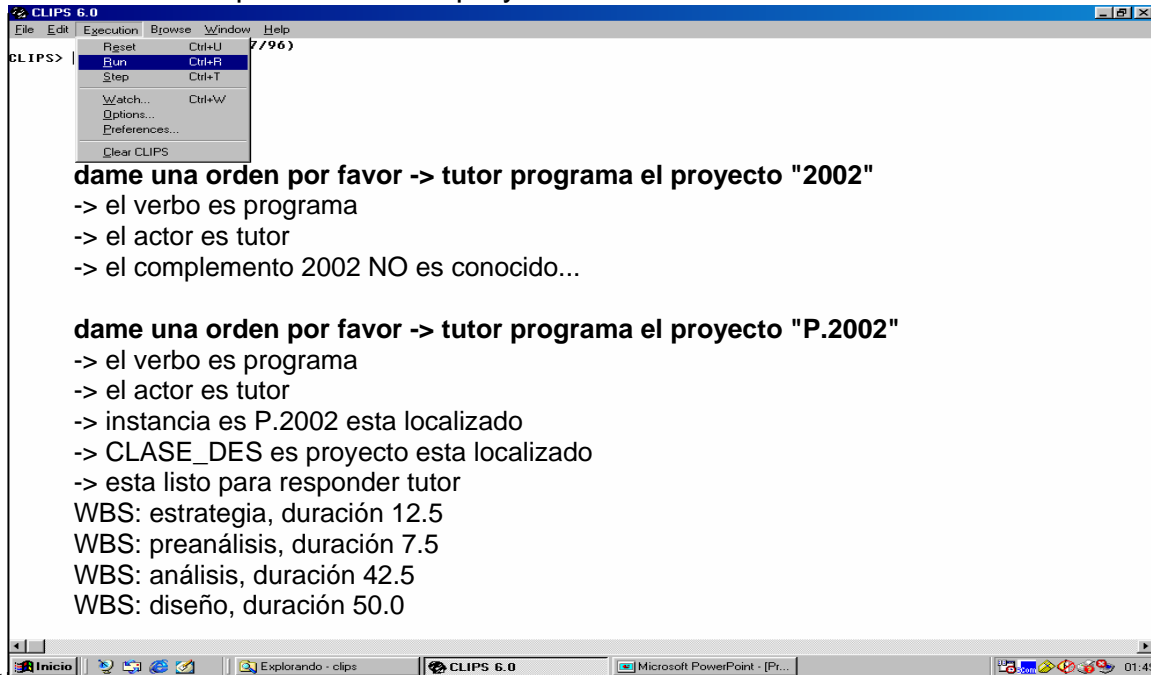
Cuando la estructura de la oración es correcta en todos sus componentes gramaticales, Clases e instancias, el SE-APIS continua de forma correcta solicitando al usuario datos adicionales del proyecto.

## Caso 3) Validar la dependencia conceptual CD programa()

La evaluación de la dependencia conceptual programa() se lleva a cabo con la declaración por parte del usuario de la oración de lenguaje natural:

“tutor programa el proyecto “P.2002”

Esta oración es analizada por el procesamiento de lenguaje natural del SE-APIS, identificando cada uno de sus componentes gramaticales y generando una respuesta correcta sobre la programación de un proyecto que ha sido generado con anterioridad. Es conveniente observar que la clave del proyecto se teclea entre comillas.



A continuación registramos la evaluación cualitativa de este escenario o grupo de pruebas para la evaluación del procesamiento del Lenguaje Natural del SE-APIS, utilizando las dependencias conceptuales reporta(), crea() y programa(), el resultado de la evaluación se reporta en el cuadro de la Figura VIII.4. Considerando como que el módulo principal en este proceso es el de *Interface.clp*.

Módulo	Interfaz con:	CD	Casos de prueba en LN	Acier-tos
Interface.clp	Métodos2.clp Intancias.clp Answers.clp Ruta_crítica	Reporta()	“tutor reporta la etapa de mantenimiento”	SI
		Crea()	“tutor crea un proyecto de metodología CADM”	SI
		Programa()	“tutor programa el proyecto “P.2002”	SI
<b>Total aciertos</b>				<b>3</b>

Figura VIII.4 Evaluación del procesamiento de Lenguaje Natural

Como puede observarse en la Figura anterior estos casos fundamentalmente permitieron evaluar el funcionamiento del procesamiento del Lenguaje Natural del SE-APIS.

Las excepciones que se encontraron asociadas con la evaluación del procesamiento del Lenguaje Natural en estos casos de prueba se listan en la Figura VIII.5.

No.	Descripción de la excepción
1	No se cuenta con la gramática para consultar y reportar etapas y actividades de diferentes metodologías, instancia-m, instancia-M, Clase-m, Clase-M

Figura VIII.5 Excepciones del procesamiento de Lenguaje Natural

#### VIII.4. Evaluación de consulta y reportes de Procedimientos y Metodologías

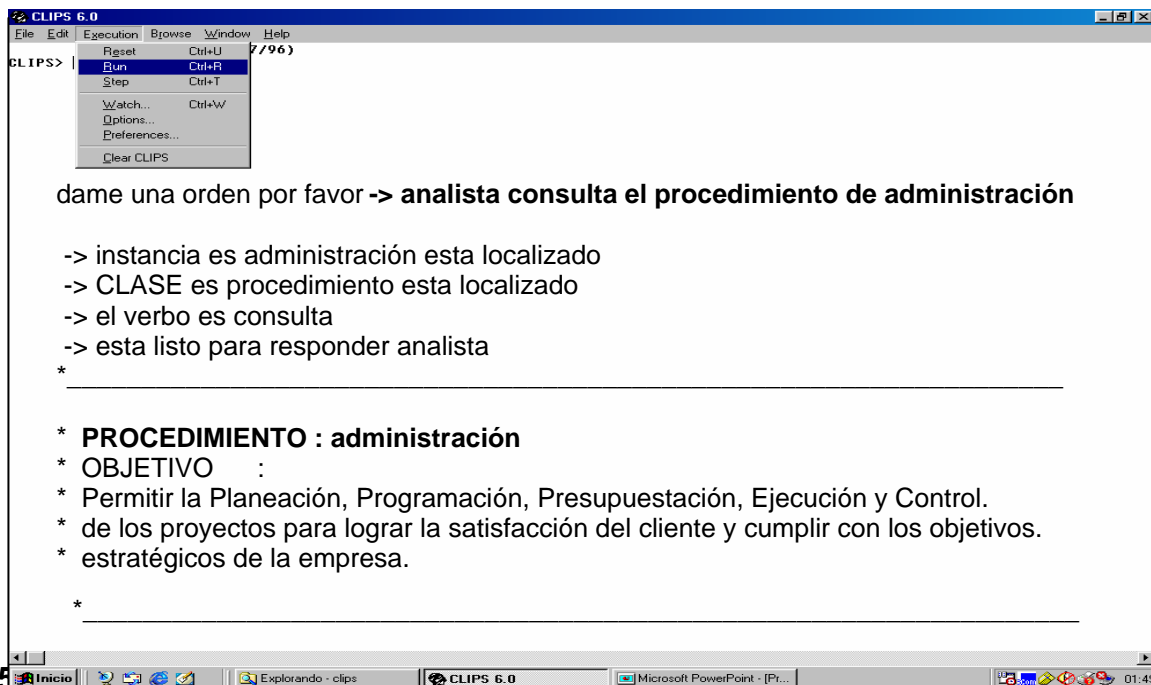
Para validar este procesamiento de conocimiento utilizaremos la funcionalidad de las CD **consulta()** y **reporta()**, para lo cual se proporcionará casos de oraciones de prueba en lenguaje natural :

##### Caso 4) Consulta sobre el procedimiento de administración

Para este caso de prueba, la oración que se le proporciona al SE-APIS en lenguaje natural es

“analista consulta el procedimiento de administración”

Con esta oración utilizando la CD **consulta()**, estamos obteniendo el conocimiento de la Clase de Nivel 1 de la estructura de *Marcos* sobre procedimientos de Administración de Proyectos y la cual puede ser consultada por parte del Ingeniero de *Software*.

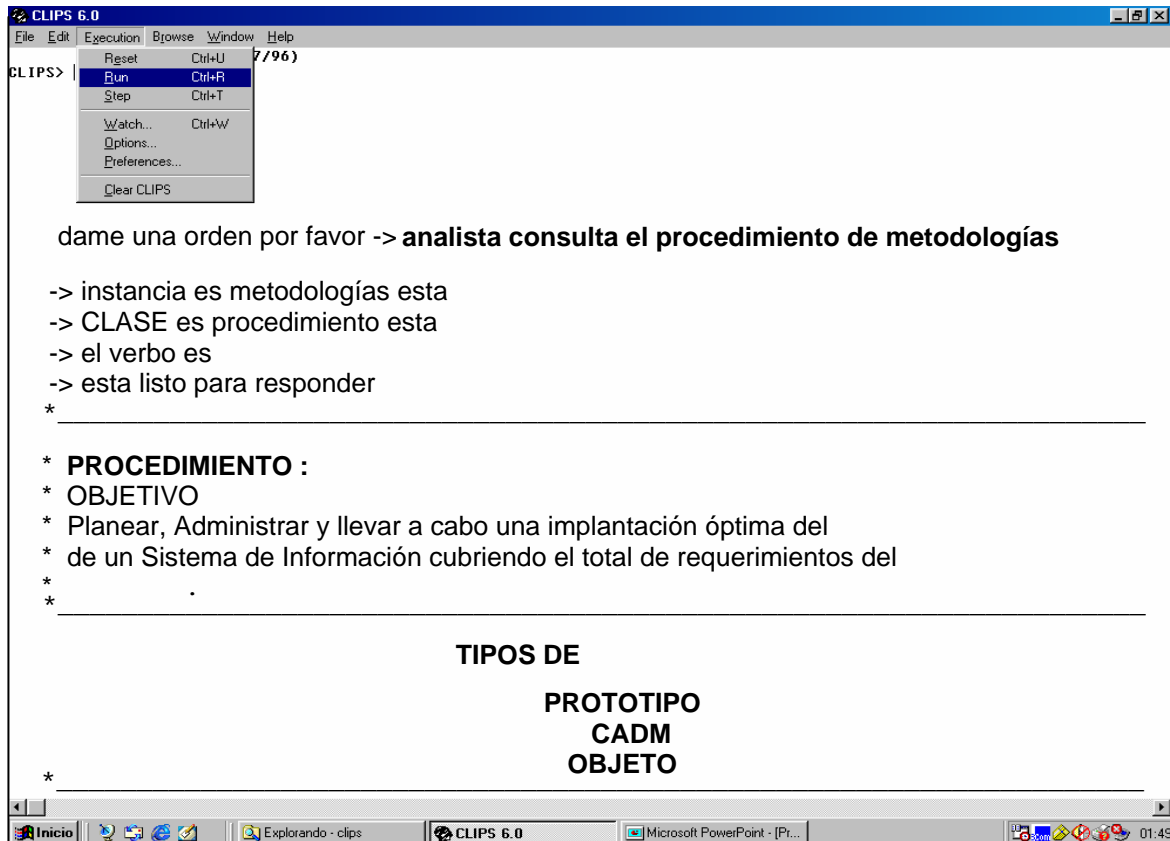


##### Caso 5)

Para este caso la oración en lenguaje natural que se le proporciona al SE-APIS es:

“analista consulta el procedimiento de metodologías”

Esta oración permite utilizar la CD consulta() y obtener información sobre la Clase de Nivel 1 del modelo de *Marcos* de conocimiento sobre procedimientos para una definición general de metodologías de desarrollo.

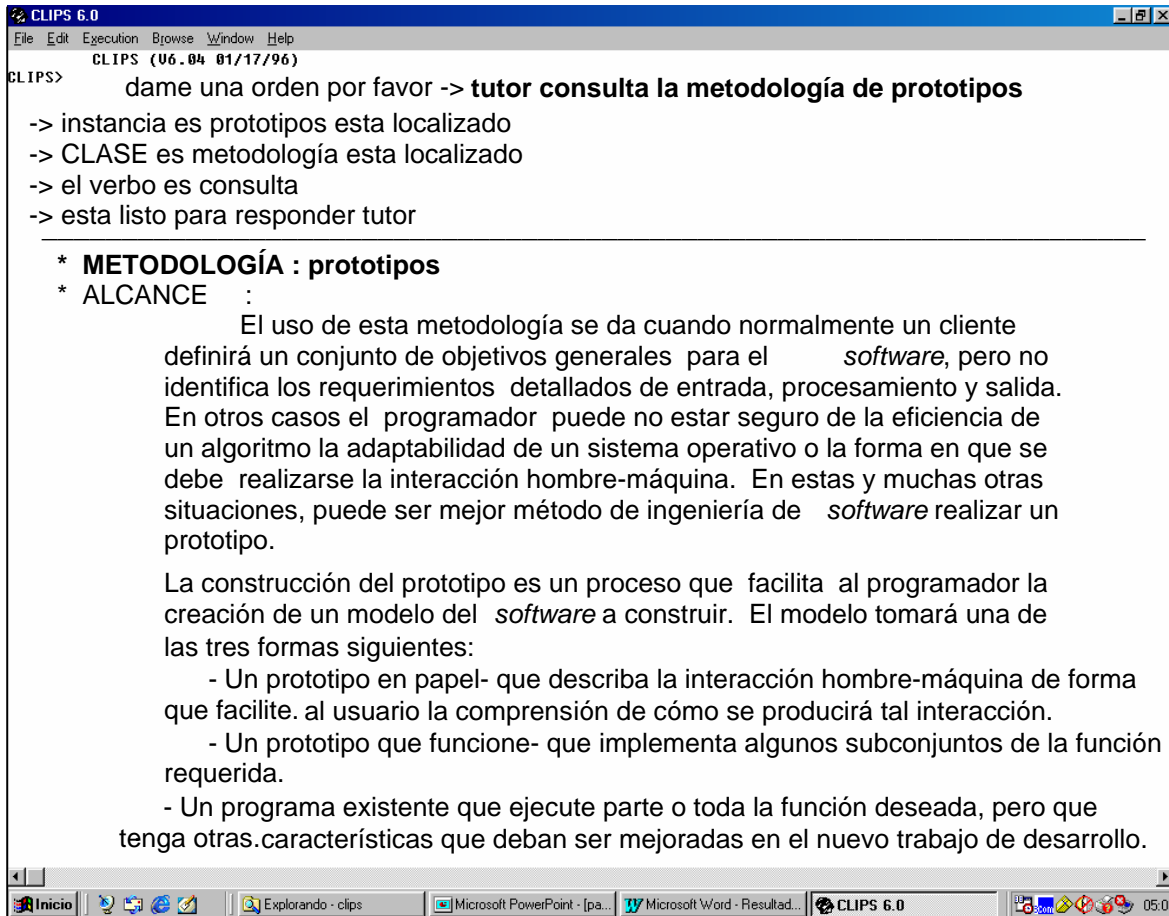


## Caso 6) Consulta sobre la metodología de prototipos

En este caso presentamos la pantalla que nos permite consultar por medio de lenguaje natural el Nivel 2 del modelo de *Marcos* de conocimiento sobre metodologías de desarrollo de *software*, utilizando la CD consulta(), con la oración tecleada por el usuario:

“tutor consulta la metodología de prototipos”

En este caso utilizamos un actor (Sintagma Nominal) diferente de acuerdo con el dominio de actores definido en el SE-APIS como “tutor”

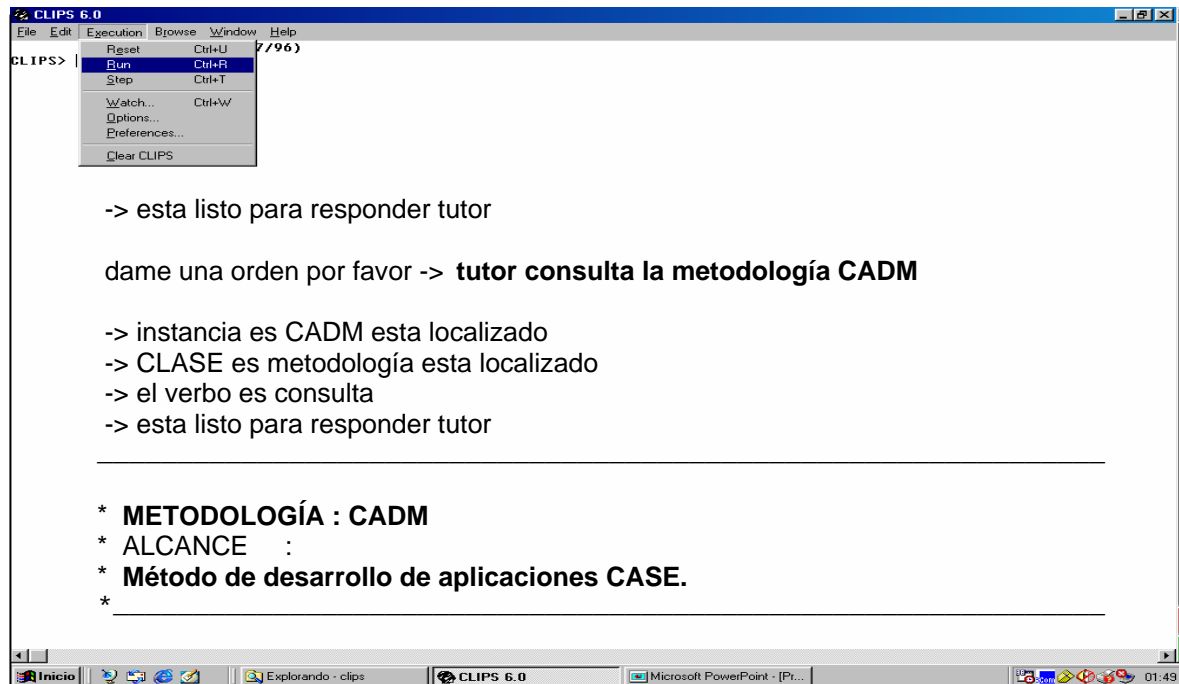


## Caso 7) Consulta de la metodología CADM

En la evaluación de este caso se utiliza la CD consulta() para obtener información sobre el Nivel 2 de la estructura de *Marcos* de metodologías de desarrollo de *software*, en forma particular de la instancia de la metodología CADM, que es un caso definido completamente en la biblioteca de CASOS listada en la tabla de la Figura IV.1, y que adicionalmente cuenta con conocimientos en los Niveles 3 y 4 para la definición de instancias de Etapas y Actividades específicas sobre esta metodología. La oración en lenguaje natural que el usuario necesita capturar dentro del SE-APIS para adquirir la definición de CADM es:



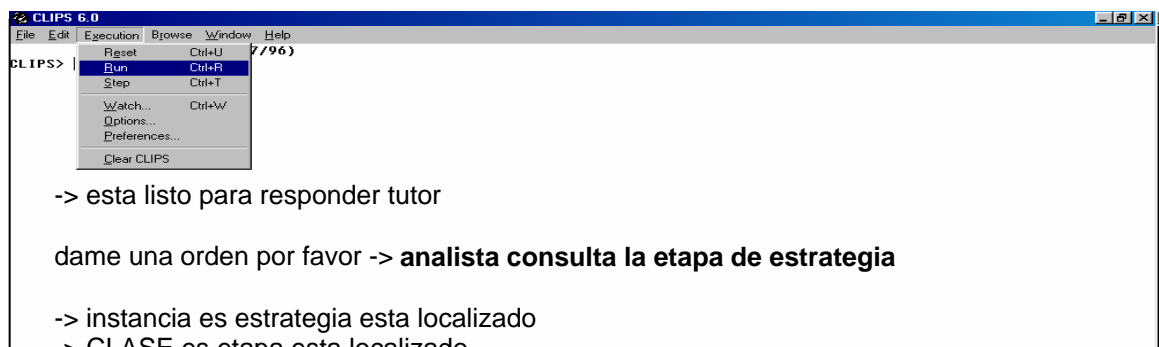
“tutor consulta la metodología CADM”



## Caso 8) Consulta de la etapa de estrategia

Para la consulta de la etapa de estrategia de la metodología CADM, se utiliza la CD consulta(), que permite obtener conocimientos sobre el Nivel 3 del modelo de Marcos de metodologías del SE-APIS, con la oración en lenguaje natural:

“analista consulta la etapa de estrategia”



\*

---

En este caso sobre la consulta de la etapa de estrategia se presenta nuevamente el problema de la excepción del Caso 1, siendo necesario contar con una estructura gramatical de la misma forma anteriormente definida.

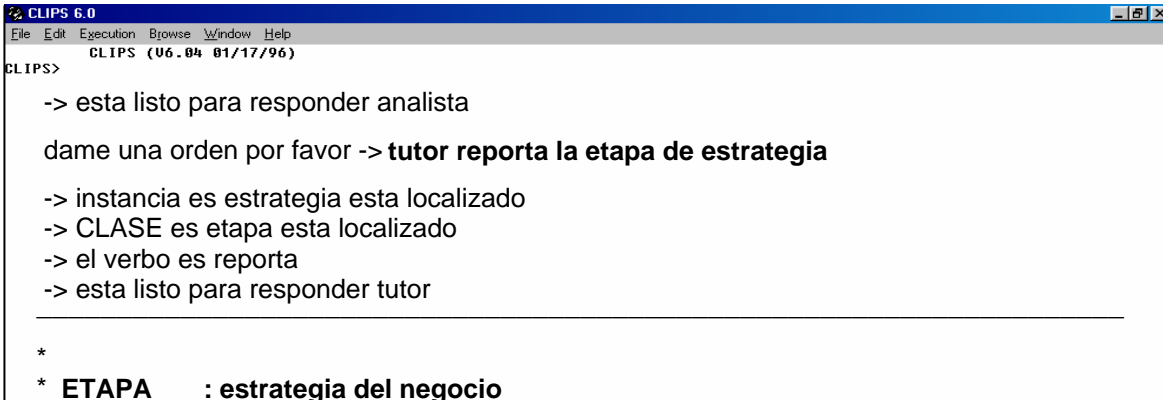
### ***SN SV artículo CLASE-m instancia-m instancia-M preposición CLASE-M***

Esta excepción del SE-APIS, no será incluida en la relación de este escenario para no repetir la que se registro en el procesamiento de Lenguaje Natural.

## **Caso 9) Reporte de la etapa de estrategia**

De manera similar al Caso 8, se requiere por parte del usuario información sobre la etapa de estrategia, pero en este caso se solicita un reporte que permita obtener la etapa de estrategia de la metodología CADM. La oración en lenguaje natural que el usuario tecleará es:

“tutor reporta la etapa de estrategia”



```
CLIPS 6.0
File Edit Execution Browse Window Help
CLIPS (U6.04 01/17/96)
CLIPS>
-> esta listo para responder analista
dame una orden por favor -> tutor reporta la etapa de estrategia
-> instancia es estrategia esta localizado
-> CLASE es etapa esta localizado
-> el verbo es reporta
-> esta listo para responder tutor
-----
*
* ETAPA : estrategia del negocio
```

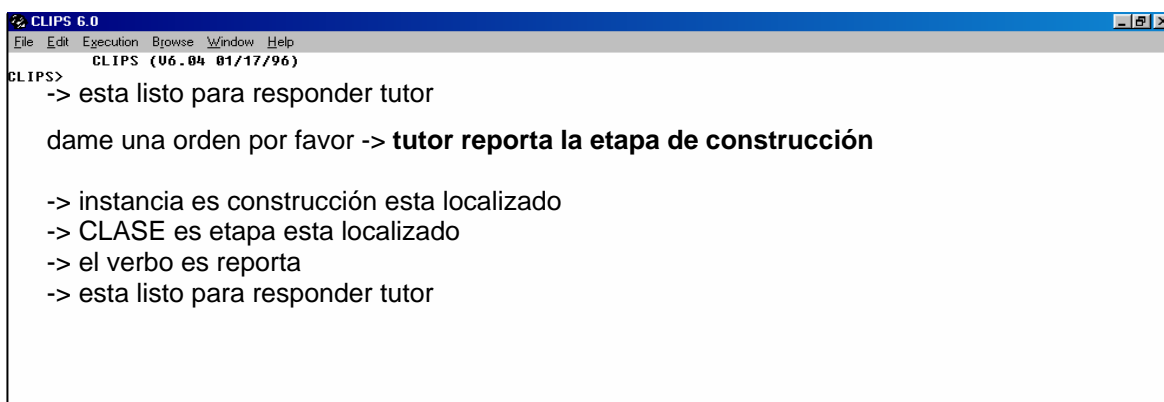
En este caso se utiliza la CD reporta() y la base de conocimientos sobre etapas y actividades que se encuentran en el Nivel 3 y 4 del modelo de *Marcos*. Los datos que se obtienen de cada actividad son: el número de la actividad, la descripción de la actividad y su % peso dentro del total de la metodología CADM.

De la misma forma que en el Caso 1, se requiere completar la gramática con una estructura en donde se incluya la instancia-m y la instancia-M, así como la Clase-m y la Clase-M, para evitar ambigüedad en caso de que se almacenen en la base de conocimientos otras metodologías.

### Caso 10) Reporte de la etapa de construcción

Este caso simplemente permite confirmar el funcionamiento del SE-APIS para reportar etapas de la metodología CADM, para lo cual se utiliza también la CD reporta(), para obtener información del modelo de *Marcos* de conocimiento del los Niveles 3 y 4. Al igual que en los casos anteriores para almacenar instancias de más metodologías se requiere incluir en la gramática instancia-m, instancia-M, Clase-m y Clase-M. La oración para obtener el reporte requerido por parte del usuario es la siguiente:

“tutor reporta la etapa de construcción”



---

\*

---

\*

---

Hasta aquí hemos concluido con los casos utilizando las dependencias conceptuales consulta() y reporta(), por lo que en este momento procederemos a la evaluación cualitativa de este grupo de pruebas utilizando el cuadro de la Figura VIII.5, para una evaluación de los módulos que participaron en este procesamiento.

Módulo	Interfaz con:	CD	Casos de prueba en LN	Acierto
Métodos2.clp	Interface.clp Intancias.clp Answers.clp	Consulta()	"analista consulta el procedimiento de administración"	SI
		Consulta()	"analista consulta el procedimiento de metodologías"	SI
		Consulta()	"tutor consulta la metodología de prototipos"	SI
		Consulta()	"tutor consulta la metodología CADM"	SI
		Consulta()	"analista consulta la etapa de estrategia"	SI
		Reporta()	"tutor reporta la etapa de estrategia"	SI
		Reporta()	"tutor reporta la etapa de construcción"	SI
<b>Total aciertos</b>				<b>7</b>

Figura VIII.6.- Evaluación de consultas y reportes de procedimientos y metodologías

Como puede observarse en la Figura anterior estos casos fundamentalmente permitieron evaluar el funcionamiento del modelo de la base de conocimientos de *Marcos*, desarrollado dentro del módulo Métodos2.clp

La lista de excepciones que se encontraron asociadas con la evaluación de estos casos de prueba se listan en la Figura VIII.7.

No.	Descripción de la excepción
1	No cuenta con un reporte de resumen de etapas de metodologías
2	No cuenta con la gramática para consultar y reportar hechos sobre proyectos generados

Figura VIII.7.- Excepciones de consultas y reportes de procedimientos y metodologías

### VIII.5. Escenario de Evaluación de Generación de Proyectos y Métricas de Putnam

Para la validación de este escenario de prueba se realizarán casos que permitan obtener una evaluación cuantitativa comparando los cálculos obtenidos por la generación de los datos del proyecto y de la métrica de PUTNAM con los cálculos elaborados fuera del SE-APIS.

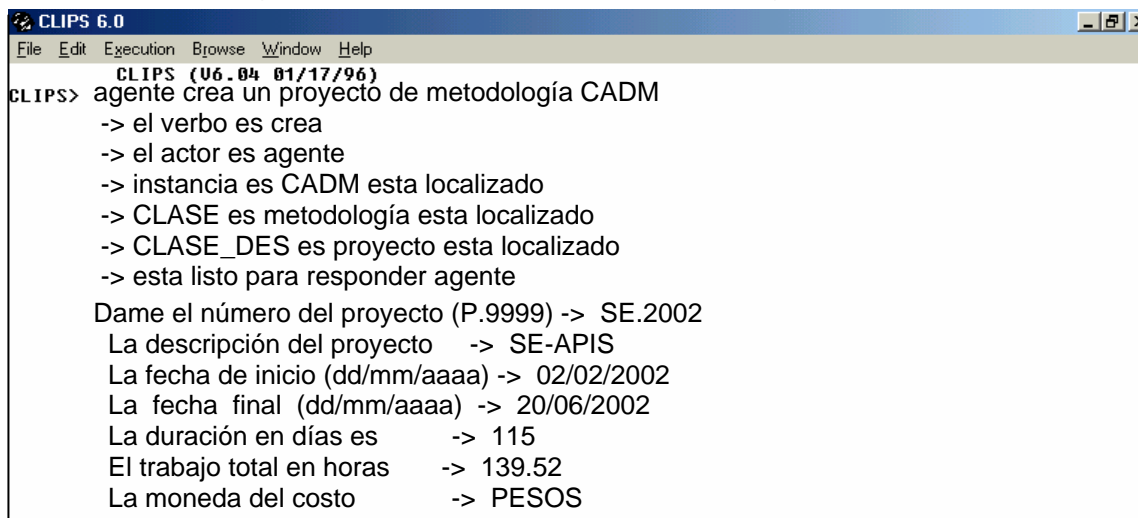
Para este escenario aplicaremos una autoevaluación del proyecto del mismo SE-APIS, aún cuando la metodología KAMET que se utilizó para desarrollar el APIS, no es totalmente compatible con la metodología CADM, que se almacena en la base de conocimientos de *Marcos*, sin embargo tomaremos en consideración datos de tres etapas similares en ambas metodologías: análisis, diseño y construcción.

#### Caso 11) Generación de un proyecto de Ingeniería de Software

Para la generación de un proyecto es necesario utilizar la CD crea(), que permita obtener información de la base de conocimientos de *Marcos* de los Niveles 2, 3 y 4 de Metodologías y utilizar los *Marcos* de los Niveles 2, 3 y 4 de Proyectos para proporcionar la estructura de ranura y relleno para la creación del proyecto, fases WBS y operaciones.

En esta CD se integró la funcionalidad tanto de la creación automática del proyecto como el cálculo de la métrica de PUTNAM. Para ejecutar este proceso es necesario que el usuario utilice la oración en lenguaje natural:

“agente crea un proyecto de metodología CADM”



Una vez que el procesamiento de lenguaje natural ha construido la CD crea(), el SE-APIS, solicita al usuario los datos globales del proyecto, entre los que se encuentran los datos numéricos: de duración, trabajo estimado en horas, costo total del proyecto y el entorno de desarrollo.

### **Evaluación de los cálculos de la métrica de Putnam**

A partir de estos datos el sistema calcula la cantidad de líneas de código de programación(LC) utilizando la siguiente expresión de las métricas del modelo de PUTNAM:

$$L = Ck \text{ (horas totales del proyecto/horas laborables * td)}^{1/3} \text{ td}^{4/3} \quad (4)$$

La cantidad de Líneas de código calculadas en el PROTOTIPO del SE-APIS en *CLIPS* da como resultado **1366.75** líneas de código comparadas contra **1179.00** líneas de código programadas realmente en el SE-APIS de acuerdo con el registro por módulos de la tabla de la Figura VII.2.

### **Evaluación de los cálculos de Horas, Costo y Duración de un Proyecto**

La dependencia conceptual crea(), permite generar automáticamente las fases WBS y las operaciones de un proyecto, incluyendo sus cálculos de horas, costos y duraciones a partir de los % en peso que contiene la base de conocimientos de cada una de ellas con respecto al 100% de las etapas y operaciones de la metodología CADM.

Para esta evaluación cuantitativa aún cuando el SE-APIS genera nueve etapas, sólo se consideran tres etapas para comparación de los cálculos obtenidos dentro y fuera del sistema.

Como ya se mencionó anteriormente las etapas de análisis, diseño y construcción son similares tanto en la metodología KAMET como en la metodología CADM. A continuación se presentan la información que el SE-APIS genera para estas tres etapas:

```

*=====
=
*
*           !!! SE GENERAN LAS ETAPAS DE DESARROLLO DEL !!!
*           SISTEMA EXPERTO COMO APOYO A LA ADMINISTRACIÓN
*           DE PROYECTOS DE INGENIERÍA DE SOFTWARE
*=====
=

```

```

.....
Se crea la ETAPA      -> (análisis del sistema)
  Con la identificación  -> 3
  Trabajo programado en horas -> 80.06999999999999
  Costo programado en PESOS -> 33150.0
  Duración programada en días -> 63.75

```

```

.....
Se crea la ETAPA      -> (diseño del sistema)
  Con la identificación  -> 5
  Trabajo programado en horas -> 94.2
  Costo programado en PESOS -> 39000.0
  Duración programada en días -> 75.0
Se crea la ETAPA      -> (construcción del sistema)
  Con la identificación  -> 6
  Trabajo programado en horas -> 146.01
  Costo programado en PESOS -> 60450.0
  Duración programada en días -> 116.25

```

Los resultados calculados dentro del SE-APIS nos permiten realizar una comparación con respecto a los datos reales totales del SE-APIS obtenidos en su construcción, considerando los % PESO estadísticos que se encuentran en las instancias de la base de conocimientos de *Marcos*.

Etapa	Datos calculados dentro del SE-APIS					Datos reales * %peso del SE-APIS			
	%Peso	Horas	Costo	Duración	LC	Horas	Costo	Duración	LC
Análisis	17	80	33,150	63.7		76.5	32,903	63.1	
Diseño	20	94	39,000	75.0		90.0	38,709	74.2	
Construcción	31	146	60,450	116.2	1366	139.5	60,000	115.0	1179
Totales	68	320	132,600	254.9	1366	306	131,612	252.3	1179
Desv.		14	988	2.6	187				

%Desv.		4.3%	0.7%	1.0%	13.7%				
--------	--	------	------	------	-------	--	--	--	--

Figura VIII.8.- Comparación de valores calculados para el SE-APIS

Los defectos encontrados en este caso de prueba básicamente se deben a que los datos que se requieren por parte del usuario para completar los hechos del proyecto, no cuentan con mascarillas de captura con reglas de validación.

Módulo	Descripción del defecto
Projects.clp	El formato de los datos solicitados al usuario no tiene una mascarilla de captura.
Total de defectos	1

Figura VIII.9.- Verificación de defectos del módulo Projects.clp

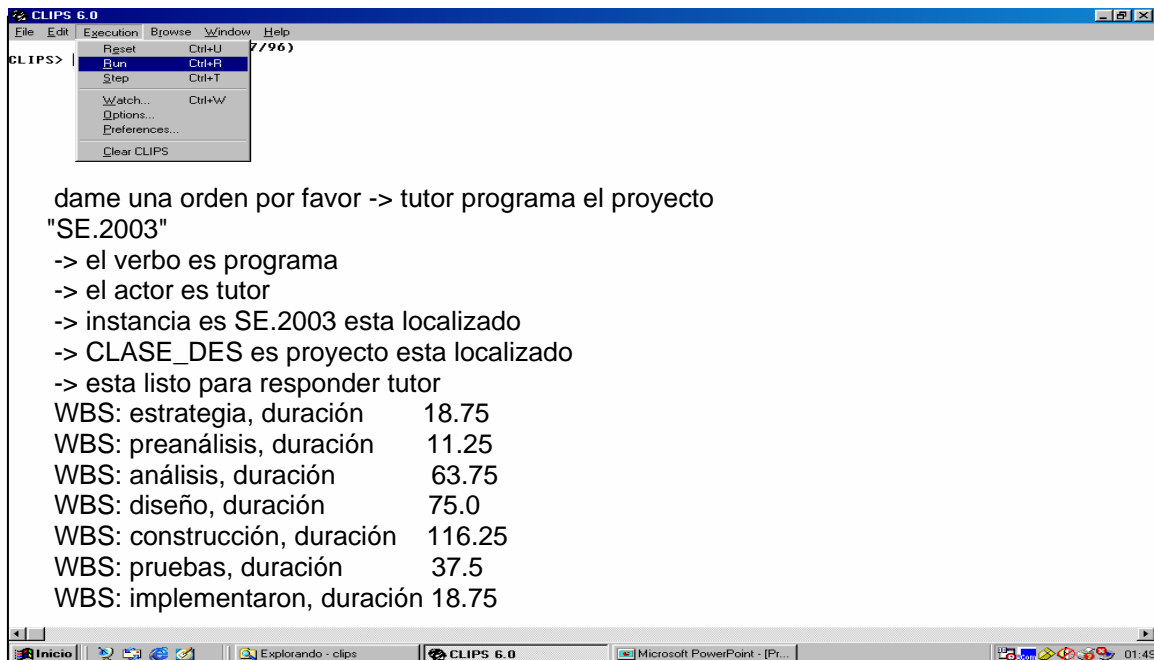
## VII.6. Escenario de Evaluación del cálculo de la ruta crítica de Proyectos

En este escenario de evaluación se realizará la programación del proyecto que previamente se genero con la CD crea(), para poder realizar esta función se ejecuta la CD programa(), con la cual el SE-APIS utiliza los hechos de duración del proyecto y los conocimientos de la estructura de Marcos de REDES que se definió en el capítulo IV de Diseño del Modelo.

### Caso 12) Cálculo de la ruta crítica de un proyecto

Dentro de este caso se ejecuta la CD programa() para poder obtener el cálculo de la ruta crítica del proyecto, que previamente ha sido creado. Para la ejecución de esta función el usuario debe dar la siguiente frase en lenguaje natural:

“ingeniero programa el proyecto “SE.2003”





En caso de que el proyecto “SE.2003” no exista el sistema contestará que la instancia es desconocida. Sí el proyecto existe el sistema obtiene los datos de las fases WBS para calcular la ruta crítica con sus duraciones. Para el caso la red de la ruta crítica del SE-APIS, se forma la que se presenta en la Figura VIII.10

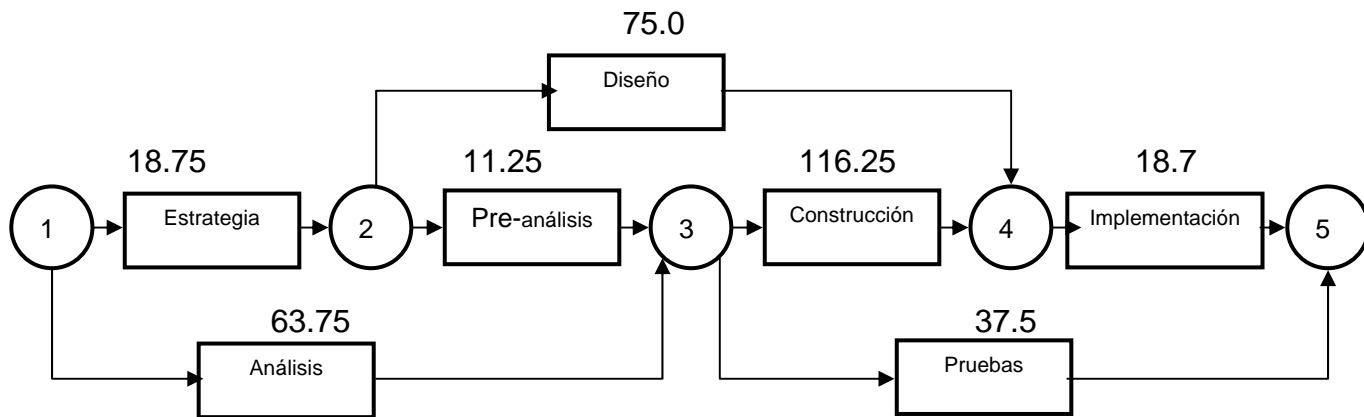


Figura VIII.10.- Red para autoevaluación del SE-APIS

Realizando una observación visual de la red de la Figura VIII.10 y un cálculo externo que acumule las duraciones mayores entre nodos, se obtiene que la ruta crítica esta formada por las fases WBS (arcos): Análisis, Construcción e Implementación y que su duración es de **198.7 días de duración**.

Los cálculos obtenidos realizando el procesamiento del diagrama de red de la Figura VIII.10 para la ruta crítica dentro del SE-APIS, para su propia evaluación son los siguientes:

**actividad, análisis de 1 a 3, duración 63.75, en la ruta crítica si**  
**actividad, construcción de 3 a 4, duración 116.25, en la ruta crítica si**

**actividad, implementación de 4 a 5, duración 18.75, en la ruta crítica si**  
 actividad, pruebas de 3 a 5, duración 37.5 , en la ruta crítica no  
 actividad, diseño de 2 a 4, duración 75.0 , en la ruta crítica no  
 actividad, preanálisis de 2 a 3, duración 11.25, en la ruta crítica no  
 actividad, estrategia de 1 a 2, duración 18.75, en la ruta crítica no

**Nodo final: 5, Duracion de la Ruta Crítica : 198.75 Unidades**

La comparación de los resultados obtenidos mediante el análisis y cálculo dentro y fuera del SE-APIS, son enteramente satisfactorios:

**/Desviación/ = Xi (fuera del SE-APIS) – Xj (dentro del SE-APIS)**  
**/Desviación/ = 198.7 – 198.75 = .05**

La única excepción que se puede establecer en este caso de prueba es que es necesario contar con redes de otras instancias de redes de conocimiento para diversas metodologías de desarrollo.

No.	Descripción de la excepción
1	No existen otras redes de instancias para diversas metodologías
Total	1

Figura VIII.11.- Excepciones del módulo de Ruta\_crítica

La realización de los doce casos de prueba para evaluar los resultados emitidos por el PROTOTIPO del SE-APIS han brindado resultados satisfactorios, de acuerdo con las tablas de evaluación cuantitativa, cualitativa, de defectos y excepciones.

## IX. Trabajos Futuros de la HIS

Considerando el amplio dominio de conocimiento que se maneja en el tema de Ingeniería de *Software*, es normal que existan mucho por desarrollar como trabajos futuros, y con mayor razón si este dominio del conocimiento lo deseamos procesar con técnicas de la Inteligencia Artificial o herramientas asistidas por computadora CASE.

### IX.1. Modelo Final de un Sistema Experto Integral HIS

En el caso particular de este trabajo de tesis, como fue mencionado en la definición del alcance, sólo se desarrollo un PROTOTIPO de Sistema Experto para apoyo a la Administración de Proyectos (APIS), quedando como trabajo futuro el desarrollo de una Herramienta Integral para la Ingeniería de *Software* (HIS).

El modelo final de la arquitectura de un Sistema Experto como Herramienta para la Ingeniería de *Software* quedo planteado en la Figura I.3 de este trabajo de tesis, y varias de sus características deseables de funcionamiento se describen en las siguientes líneas:

- Adquisición de Conocimientos.- Permitiendo desde una simple introducción directa de conocimiento de metodologías y administración de proyectos por parte del usuario así como aprendizaje memorístico (Samuel, 1963 [SAM63]), hasta un ajuste de parámetros y coeficientes de métricas y estadísticas para desarrollo de *software* por medio de aprendizaje autónomo a partir de los registros reales de tiempos, recursos y costos en el desarrollo de Proyectos y Sistemas realizados por el usuario con metodologías de *software* o con la incorporación de herramientas CASE.
- Base de Conocimientos de Metodologías.- Desarrolladas por medio de *Marcos* (Minsky 1975, [MIN75]) y dependencias conceptuales CD (Schank, 1975[SHA75]) para almacenar metodologías de Ingeniería de *Software* que incluyan información de la definición propia, definiciones de sus diferentes etapas y de cada una de sus actividades, asimismo cada actividad debe incluir pesos estadísticas de tiempo, recursos y costos relativos a la propia actividad dentro de la metodología. Esta base de conocimientos debe actualizarse por medio del módulo de adquisición del conocimiento.
- Bases de Conocimientos de Métodos, Modelos y Métricas.- Esta base de conocimiento debe estar relacionada a la de metodologías para aportar detalles de Modelos E-R, de Procesos, de Objetos y Funcionales, así como Métodos por ejemplo de interfaces, algoritmos específicos y de análisis de recopilación de requerimientos de usuarios, adicionalmente el conocimiento almacenado sobre métricas permitirá junto con las estadísticas de la metodología obtener con mayor precisión cálculos de tiempo, recursos y costo para los proyectos de Ingeniería de *Software* a desarrollar, entre estos métodos de métricas se encuentran el de PUTNAM y el COCOMO 2 (Bohem, 2000 [BOE00]).
- Base de Conocimientos de Proyectos de *Software*.- La estructura de esta base de conocimientos será utilizando *Marcos* y CD, representada en diferentes niveles desde el WBS del proyecto, sus etapas, operaciones y recursos tal como lo define la PMI (Project

Management Institute [PMI96]) se encontrará relacionada con una base de datos para el almacenamiento de instancias de los proyectos que se encuentren en desarrollo. Asimismo se relacionará con las bases de conocimientos de las metodologías, modelos, métodos y métricas para permitir la creación automática de proyectos de Ingeniería de *Software*. Esta base de conocimientos incluirá reglas de conocimiento que ayuden al Líder del proyecto a administrar el conjunto de etapas del ciclo de vida del proyecto: planeación, programación, ejecución, control, evaluación y cierre.

- Base de Conocimientos de Desarrollos de *Software* y Sistemas.- Esta base de conocimientos tendrá almacenada la información de desarrollo del *software* o Sistema de Información que se esté realizando, entre otros conocimientos tendrá la documentación de los modelos, métodos, lenguajes y documentación del *software* desarrollado que se está administrando desde el proyecto. Las mediciones reales de sus parámetros de tiempo, recursos, e información de desarrollo como por ejemplo cantidad de tablas y atributos en la base de datos, pantallas de interfaces con el usuario, líneas de código, procesos, reportes y programas de consulta serán evaluados para incorporarlos como estadísticas del Sistema Experto por medio del módulo de adquisición de conocimiento [RPE02]. Este módulo permitirá establecer una interfaz con las herramientas CASE que se estén utilizando para el desarrollo del Sistema de Información.
- Analizador de parámetros y coeficientes estadísticos.- Este módulo permitirá analizar, clasificar y evaluar los parámetros y estadísticas surgidos de la Administración de Proyectos y de los Sistemas de Información desarrollados, se formarán grupos de parámetros más realistas que se ocuparán en la actualización del conocimiento del Sistema Experto y que permitan generar de forma automática propuestas para los proyectos futuros de Ingeniería de *Software* de las empresas. El diagnóstico y los resultados obtenidos en este módulo serán incluidos en el Sistema Experto por medio del módulo de Adquisición de Conocimiento.
- Interfaz con el usuario.- La interfaz con el usuario debe cumplir las premisas de ser sencilla, flexible y eficaz por lo que se desarrollará con varios elementos: Un procesador de Lenguaje Natural con gramática de contexto libre y una programación orientada a objetos para el desarrollo de pantallas y menús de selección de funciones para el usuario. Estos deben de operar en un ambiente de Internet para incrementar su potencialidad de difusión de conocimientos y de desarrollo concurrente [JES02].
- Reglas de Inferencia de Proceso y Control.- El desarrollo de este conjunto de reglas dentro de un motor de inferencia permitirá contar con procesos más eficientes y con una secuencia de razonamiento inteligente soportado por heurísticas y métodos de Inteligencia Artificial para realizar las búsquedas de conocimiento y su procesamiento ordenado dentro del Sistema Experto para Integral para la Ingeniería de *Software* [KEK94] y [SRU96].

## IX.2. Incorporación de métricas de desarrollo de *Software*

Las métricas del *Software* son fundamentales en los Proyectos de Ingeniería de *Software*, es por esta razón que en el PROTOTIPO desarrollado del SE-APIS, se incluyó de forma básica el modelo de la ecuación de PUTNAM, que permite obtener el esfuerzo necesario en personas/mes u horas requeridas para programar una determinada cantidad de líneas de código (LC) de un sistema de información o *software* a construir.

No obstante el desarrollo del modelo de PUTNAM dentro del SE-APIS, es necesario desarrollar como trabajo futuro, dentro de los módulos del SE-HIS, un mayor número de métricas, dentro de las cuales se pueden incluir:

- Métricas orientadas a la Función y Complejidad del *software*
- Métricas para la estimación del Costo del *software*
- Modelos para la Calidad y Análisis de Riesgo

Aún cuando existen actualmente herramientas CASE con métricas, estas no están completamente integradas en una Herramienta de Ingeniería de *Software*.

Dentro de las características deseables que las métricas deben cumplir para integrarse en un trabajo futuro dentro del HIS, se encuentran las siguientes:

- Métricas orientadas a la Función [ALB79], con las que a partir de medidas de Puntos de Función, se pueden obtener mediciones contables según el dominio de la información del *software* y su complejidad (simple, media o alta), entre otras mediciones se pueden considerar:
  - Número de entradas del usuario
  - Número de salidas del usuario
  - Número de peticiones del usuario
  - Número de archivos
  - Número de interfaces externas
- Métricas para la estimación del costo, como por ejemplo el Modelo de Barry Boehm, llamado COCOMO II [BOE00], incorporando un grupo de modelos de estimación para tratar las siguientes modelos:
  - Modelo de la composición de la aplicación.- interfaces de usuario, evaluación del rendimiento, y evaluación de la madurez de la tecnología.
  - Modelo de la fase del diseño previo.- que incluya métodos para la recolección de requerimientos de desarrollo del *software*.
  - Modelo de la fase posterior a la arquitectura.- Utilizado durante la construcción del *software*.

- La incorporación de métricas en el HIS, debe incluir otras no menos importantes tales como son el de la medición de la calidad del *software* y análisis de riesgo del desarrollo de proyectos de Ingeniería de *Software*.

### IX.3. Desarrollo en ambiente Internet

Dada la importancia de una Herramienta Integral para la Ingeniería de *Software* HIS, sería de gran valor como trabajo futuro, su desarrollo en un ambiente de procesamiento cooperativo y de fácil difusión como lo es Internet.

Existen motores de inferencia para Sistemas Expertos como JESS (Java Expert System Shell [JES02]) desarrollado por Friedman, que proporciona un excelente ambiente de desarrollo para este tipo de sistemas inteligentes. JESS está completamente escrito en el lenguaje de programación Java, soporta reglas de inferencia en *CLIPS*, que pueden acoplarse de una manera sencilla.

JESS reúne varias condiciones necesarias para hacer de una HIS, una potente herramienta para los Ingenieros de *Software* en un ambiente Internet. Dentro de las características deseables para un desarrollo futuro de una HIS en este ambiente están:

- Que se cuente con un motor de inferencia para la activación de reglas que conocimiento, representadas con conocimiento heurístico de expertos humanos.
- Permita desarrollar un amplio rango de reglas para razonar en forma de *agentes inteligentes* en conjunto con sistemas ERP (*Enterprise Resource Planninig*), por ejemplo con *software* como SAP [PSP99].
- Procesamiento para comercio electrónico.
- Proporcione un lenguaje de programación potente que maneje directamente rutinas de Java y sus librerías, y *API*'s.
- Manejo de *Applets* para una sencilla comunicación con aplicaciones en Internet y permita el desarrollo de aplicaciones WEB.

Los tres apartados sobre trabajos futuros descritos en este capítulo pueden desarrollarse de forma integral para construir un potente Sistema Experto como Herramienta Integral para la Ingeniería de *Software* HIS, dentro de un trabajo de investigación de actualidad.

## Conclusiones

El prototipo SE-APIS del Sistema Basado en Conocimiento como Herramienta de apoyo para la Administración de Proyectos de Ingeniería de *Software* HIS, es un PROTOTIPO de herramienta inteligente que puede apoyar a los Ingenieros de esta especialidad en el conocimiento y estandarización de metodologías de desarrollo de sistemas y la Administración de Proyectos de *Software*.

Con la implantación del PROTOTIPO SE-APIS es posible apoyar en la solución de varios de los problemas detectados en la Administración de Proyectos y manejo de metodologías. Dentro de las mejoras que se es posible lograr con su incorporación están las siguientes:

- Mejorar la planeación de los proyectos.
- Contar con una metodología en línea que guíe en el desarrollo de *software*.
- Proporcionar entrenamiento en línea sobre metodologías y Administración de Proyectos de *Software*.
- Obtención de estimados de costo, recursos y duración de las diferentes etapas y actividades del proyecto.
- Identificar las actividades críticas del proyecto.
- Contar con estadísticas mínimas para el desarrollo de *software*.
- Incremento gradual del conocimiento del SE-APIS, con la incorporación de un mayor número de casos de metodologías en su biblioteca.

De acuerdo con él “Estado del Arte” actual sobre los diversos productos y soluciones que apoyan a la Administración de Proyectos y desarrollo de *software*, se observa que aún cuando existen varias soluciones que apoyan esta actividad, es necesario realizar una herramienta que apoye a los Ingenieros de *Software* de manera integral.

Los conceptos de la Ingeniería de *Software* actuales, forman un amplio e interesante dominio de conocimiento, que es factible de ser incorporado en un Sistema Experto Integral, manejando diversas técnicas y tecnologías de la Inteligencia Artificial.

El diseño del Modelo del SE-APIS se facilitó gracias a las técnicas de la Inteligencia Artificial establecidas y dentro de las cuales podemos mencionar las siguientes:

- El uso de estructuras de ranura y relleno débil Marcos u Objetos se adaptó de una forma natural al tipo de conocimiento de Metodologías de Ingeniería de *Software* y Proyectos para sistemas de Información por lo que fue posible aprovechar las propiedades de jerarquía de clases, herencia y mensajes entre objetos.
- La interfaz basada en Lenguaje Natural facilitó el diseño de la Interfaz con el usuario, sin embargo aún existe ambigüedad en la interpretación semántica de las frases.
- La representación del conocimiento relacional simple facilitó el diseño de la generación y almacenamiento de hechos de las estructuras de los Proyectos.

- El manejo de reglas de inferencia con CD apoyó el diseño del procesamiento de los diferentes analizadores y controles del motor de inferencia, la creación y programación de proyectos, el cálculo de parámetros: como el costo, horas y duración con los pesos estadísticos de las fases y operaciones de las metodologías, así como el cálculo de las líneas de código con el método de Putnam.

La construcción del PROTOTIPO del SE-APIS en el *software CLIPS (C Language Integrated Production System)* se realizó de una manera relativamente sencilla, por lo que es una motivación a continuar con el desarrollo de este tipo de Sistemas Inteligentes para aplicarlos en el IMP.

Los resultados obtenidos sobre la evaluación cuantitativa y cualitativa del SE-APIS fueron satisfactorios, permitiendo obtener una constancia de que el objetivo y el alcance planteado se cumplieron ampliamente, bajo los siguientes escenarios de prueba:

- Interfaz de Lenguaje Natural.
- Consulta y reportes de Procedimientos y Metodologías.
- Generación de Proyectos y Métricas de Putnam.
- Cálculo de la ruta crítica de Proyectos.

El desarrollo de trabajos futuros para construir una Sistema Basado en Conocimiento como Herramienta Integral de apoyo a la Ingeniería de *Software HIS*, a partir del PROTOTIPO del **SE-APIS**, tiene una gran posibilidad de desarrollarse en un ambiente Internet colaborativo como **JESS** y de poder incorporar un servidor de Base de Datos que permita realizar análisis sobre los proyectos de *software* y sistemas de información soportado por una Base de Conocimientos que incluya **Métricas de Software**, con lo que se obtendrá una Herramienta que apoye fuertemente a los Líderes de Proyecto de *Software* en el IMP.



## ANEXO B

### **I.2.- Método de desarrollo de aplicaciones CASE (CADM).**

Un proyecto mal planificado se realizará en cinco veces el tiempo previsto. Uno bien planificado en sólo tres veces el tiempo previsto.

La clave para el éxito de un proyecto de diseño de sistemas radica en la coordinación de diversos factores críticos para el éxito: personas adecuadas, tecnología/herramientas adecuadas y método adecuado. Hace falta una combinación efectiva de estos tres factores para el éxito del sistema.

#### **I.2.1.- Panorámica del método de desarrollo de aplicaciones CASE (CADM)**

Todo ciclo de vida del desarrollo de sistemas (SDLC, System Development Life Cycle) se reduce, en esencia, a que todas las metodologías dirigen una estructura para planificar, analizar, diseñar, construir e implementar aplicaciones de negocios: Lo que las diferencia es su nivel de detalle y técnicas.

El Método Barker tiene las fases básicas de estrategia, análisis, diseño, construcción, documentación, transición y producción. Se han añadido a la metodología básica los pasos de transición preanálisis y prediseño, e incluido implementación y mantenimiento en el conjunto de todo el proceso. A este método revisado se le denomina método de desarrollo de aplicaciones **CASE CADM** (*CASE Application Development Method*).

#### **I.2.2.- Estrategia**

La fase de estrategia está totalmente centrada en el negocio. El objeto de la fase de estrategia es conseguir una buena comprensión de las metas de cada área, objetivos, procesos, dirección y necesidades del negocio para estructurar y documentar la visión de un proyecto

El documento de estrategia perfila el alcance de un proyecto y define un acuerdo sobre qué debe proporcionar el proyecto. También incluye el presupuesto estimado, estructura de tiempos, recursos y estándares con los que el proyecto debe completarse. Es la base para avanzar y preparar los planes de detalle.

La Tabla 1.1 muestra una lista de las fases básicas del método CASE de desarrollo y qué tareas son compatibles en cada una.

Fases de diseño CASE revisado	Permitido por CASE comercial	No permitido por la Versión actual de CASE comercial	Puede ser permitido por CASE comercial con Extensibilidad de usuario
Estrategia	Diagrama E/R de estrategia Flujos de procesos	Documentos de Estrategia Análisis de coste- Beneficios Plan de trabajo	
Análisis	Diagrama E/R de análisis Jerarquía de Funciones Diagramas de flujos de datos	Entrevistas al usuario Prototipos Guiones gráficos	Requisitos de Sistema Revisión de informes
Diseño	Módulos básicos	Módulos complejos	Especificaciones de diseño
Construcción Pruebas	Módulos básicos	Módulos complejos Pruebas Automatizadas	Plan de pruebas y Ejecución
Implementación	Plan de Implementación		
Mantenimiento	Gestión de versiones		

Barker establece muy claramente que el objetivo de la fase de estrategia es producir, bajo la dirección del usuario, un conjunto de modelos de negocio, un conjunto de recomendaciones y un plan acordado para el desarrollo de sistemas de información.

Una vez completada la fase de estrategia, es importante entregar un documento de estrategia de alta calidad que incluya las siguientes partes: un diagrama entidad relación (diagrama E/R) de estrategia, un documento de requisitos de alto nivel, un análisis del entorno político, un plan de los flujos de trabajo, flujos de procesos a nivel estratégico y una evaluación de la estrategia.

## **Diagrama E/R de estrategia**

El propósito del diagrama entidad relación de estrategia es identificar los datos primarios o entidades principales del área de negocios. El diagrama entidad relación de estrategia sirve para demostrar una comprensión preliminar del área de negocios. Los diagramas entidad relación de estrategia deben centrarse más en la legibilidad que en la teoría relacional.

## **Documento de estrategia**

El objetivo principal de este documento es delimitar el alcance del proyecto; identificar y describir los requisitos de alto nivel. Otro objetivo es vender el sistema. Se debe incluir un análisis de coste-beneficios.

## **Respaldo del negocio y formal**

El análisis de sistemas tiene que tener cuidado con el entorno político de la organización en la que se esté trabajando, puede tener un gran impacto sobre cómo se desarrollen las fases del proyecto. Este entorno debe ser documentado.

## **Plan de trabajo de alto nivel**

Un plan de trabajo de alto nivel identifica las principales fases, actividades, hitos, entregas clave, recursos y duración del proyecto.

## **Flujos de procesos a nivel estratégico**

Los procesos clave del negocio deben ser identificados y modelizados.

## **Evaluación estratégica**

La evaluación de la estrategia requiere la participación tanto de usuarios como de desarrolladores, es importante educar a los usuarios hasta el punto de que puedan evaluar con efectividad la estrategia del proyecto.

Un documento de estrategia es, en esencia, un contrato que debe hacer una coordinación entre las partes. Los desarrolladores deben comprender las necesidades y dirección del negocio; y los usuarios deben tener confianza en que los desarrolladores han alcanzado el suficiente nivel de comprensión como para seguir adelante.

### **I.2.3.- Preanálisis**

Los objetivos del preanálisis son planificar el proceso de análisis y comenzar a fijar los estándares. Usuarios y desarrolladores, en conjunto, deben desarrollar una estrategia que asegure que el análisis se realizará correctamente.

El documento de preanálisis debe incluir: el plan de análisis, plan para implementar los estándares CASE de análisis y evaluación del preanálisis.

#### **Plan de análisis**

Antes de que el análisis pueda comenzar, debe hacer un plan de análisis. Se deben incluir pasos y entregas, debe incluirse una explicación del alcance del análisis.

#### **Plan para implementación de los estándares CASE de análisis**

El desarrollador debe especificar cómo van a ser implementados los estándares CASE DE ANÁLISIS. ¿Cómo se usará la herramienta CASE? ¿Qué campos se rellenarán y con qué información? ¿Cómo se llamarán las entidades y los atributos? ¿Qué grado de detalle tendrán las descripciones? ¿Las funciones se numerarán o se les proporcionará un nombre? ¿Qué se ignorará? Estas son algunas de las muchas preguntas que deben ser contestadas antes de continuar con el diseño.

#### **Evaluación del preanálisis**

El preanálisis estará terminado cuando todos los requisitos del sistema se cumplan, para lo cual debe realizarse una evaluación del mismo.

### **I.2.4.- Análisis**

El objetivo de la fase de análisis consiste en obtener todas las especificaciones de usuario para el proyecto y detallar completamente todos los procesos del negocio que se verán implicados.

El diagrama entidad relación y la jerarquía funcional se construyen, directamente, con el usuario e, indirectamente, mediante el documento de requisitos, la comunicación con el usuario debe ser continua.

Los procesos del negocio existentes, el sistema heredado y los usuarios mismos son todas las fuentes de información de requisitos.

El documento de análisis consta de las siguientes partes: diagrama entidad relación de análisis, flujos de procesos (lógicos), documento de requisitos y evaluación del análisis.

### **Diagrama E-R de análisis**

El objetivo del diagrama entidad relación de análisis es representar completamente tantas reglas del negocio como sea posible. Cualquier regla que no pueda ser representada en el modelo debe hacerse constar por escrito.

### **Flujos de procesos (lógicos)**

El desarrollador necesita documentar los flujos de procesos lógicos. ¿Cuáles son las tareas? ¿Quién las llevará a cabo? ¿Qué tareas preceden a otras? Los flujos de procesos modelan las principales operaciones del negocio intentando responder a estas preguntas, con un cuadro de flujos por cada operación.

### **Documento de requisitos**

El documento de requisitos es una parte crítica de la fase de análisis. Cuando se trabaja con un sistema heredado, un documento de requisitos debe incluir lo siguiente:

- Objetivos de negocio detallados y factores críticos para el éxito por cada área del negocio incluida en el sistema.
- Documentación del sistema heredado debe modelizarse y construir una pequeña base de datos que contenga la referencia de las tablas, campos, aplicaciones y pantallas de menú existentes, siendo de gran ayuda para estar seguros de que el nuevo sistema no perderá ninguna funcionalidad del sistema heredado.
- Lista de requisitos. Las entrevistas a los usuarios deben ser estructuradas de tal manera que los requisitos sean fácilmente identificados y asignados a áreas funcionales. Cada entrevista debe generar un documento que se le devuelve al usuario para que dé su visto bueno.
- Una revisión de los informes.
- Flujos de procesos para las principales funciones del negocio.

- Jerarquía de funciones. Un análisis de la jerarquía de funciones debería recoger las funciones básicas del negocio.
- Reglas del negocio. Se representan en el diagrama entidad relación de análisis. Sin embargo, hay reglas del negocio cuyo nivel de datos ni siquiera permite su representación en un diagrama entidad relación. Por lo tanto, esos requisitos tienen que ser localizables en el texto y, en la fase de construcción, traducidos a disparadores de la base de datos.
- Requisitos referenciando la jerarquía de funciones. Para probar la exactitud de la jerarquía de funciones, todos los requisitos de la lista detalladas de requisitos tienen que hacer referencia a la jerarquía de funciones. Tendrán que volver a ser encaminadas, y probablemente eliminadas, aquellas funciones que no representen una necesidad del usuario. La creación de nuevas funciones será necesaria en el caso de aquellos requisitos que no puedan hacer referencia a funciones existentes.

El documento obtenido se puede usar para hacer una comprobación cruzada, asegurándose de que los modelos de datos y funciones propuestas están completas. Un documento de requisitos de este tipo incrementa enormemente las probabilidades de éxito del diseño del sistema.

El documento de requisitos debe incluir lo siguientes: jerarquía de funciones de análisis, documentación del sistema heredado, reglas del negocio y requisitos referenciando la jerarquía de funciones.

## **Evaluación del análisis**

La primera pregunta de la evaluación del análisis es si el conjunto de requisitos es completo. Si se ordenan los requisitos por funciones detalladas y se muestran a los usuarios, éstos suelen encontrar algo nuevo en ese momento. Finalmente, corresponde a los desarrolladores asegurarse de que los requisitos a nivel de sistema han sido concretados.

La siguiente pregunta es si el diagrama entidad relación es correcto. Un diagrama entidad relación lógico correcto es una representación en tercera forma normal de tantas reglas del negocio en forma de dato/relación como sean posibles. Cada una de las reglas que no pueda modelizarse se debe incluir como una especificación de disparador que acompañe al diagrama entidad relación.

La pregunta final de la evaluación del análisis es si la jerarquía de funciones es correcta. El propósito de la jerarquía de funciones es mostrar los procesos del negocio y asegurarse de que todos los requisitos del sistema están planificados para su implementación. La jerarquía de

funciones se utiliza principalmente con propósitos organizativos. Si todas las principales funciones del negocio están representadas, todos los requisitos están haciendo referencia al menos a una función y cada función tiene al menos un requisito asociado con ella, entonces la evaluación del análisis ha finalizado.

## Reingeniería de procesos del negocio

La reingeniería de procesos del negocio (BPR) es un rediseño radical del modo subyacente en el que una organización realiza sus tareas.

Normalmente, cuando se rediseña un sistema es una buena oportunidad para no sólo reemplazar el sistema heredado actual o automatizar un proceso manual, sino para pensar sobre qué cambios pueden hacerse a los procesos subyacentes del negocio para hacerlos más rápidos, más eficientes y que satisfagan mejor las necesidades de la organización. La reingeniería de procesos del negocio no es una de las fases del proceso de desarrollo de sistemas tradicional, pero se puede añadir para ensanchar el alcance del sistema propuesto.

Katherine Duliba ha estudiado la forma en que las compañías realizan la reingeniería en sus negocios, desarrollando la taxonomía mostrada en la Tabla 1.2. Observó que cuando se construye un nuevo sistema, la reingeniería no consiste en una proposición de todo o nada. Existen grados de reingeniería que reflejan la cantidad y tipos de los cambios efectuados al sistema existente. En la tabla, las "X"s representan lo que está siendo sometido a un proceso de reingeniería en cada paso.

Las columnas nos proporcionan una lista de los elementos que pueden cambiar en el proceso de reingeniería:

**Tabla 1.2.** Taxonomía de la reingeniería de procesos del negocio

	<b>Proceso</b>	<b>Software</b>	<b>Interfaz</b>	<b>Hardware</b>	<b>Datos</b>	<b>Personas</b>
Reafirmación						
Reembalaje			<b>X</b>			
Rehost				<b>X</b>		
Rearquitectura		<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	
Reingeniería	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>	<b>X</b>

Las filas de la tabla indican el grado de cambio, desde el más pequeño hasta el más radical:

- **Reafirmación.** Supone una decisión explícita de no hacer nada, alternativa que siempre debe ser considerada.

- **Reembalaje.** Cambia la interfaz de usuario, por ejemplo, desde una interfaz de tipo carácter a una interfaz GUI o basada en Web.
- **Rehost.** Cambia fundamentalmente la plataforma hardware, por ejemplo, migrando desde un entorno centralizado en un mainframe a un entorno cliente/servidor.
- **Rearquitectura.** Cambia todo el núcleo tecnológico sin afectar explícitamente a la forma en que se realiza el negocio.
- **Reingeniería.** Cambian todos los factores arriba mencionados desde el sistema antiguo al nuevo.

También es necesario observar el alcance del cambio en la organización. Los cambios pueden ser implementados a nivel individual, departamental, de división u organizacional. Cuanto mayor sea el cambio, y mayor su alcance, mayores serán los potenciales beneficios y mayores los riesgos potenciales. Cambios mayores tienen costes iniciales mayores y resultados inmediatos menores, con un ciclo de vida más largo. Cambios menos severos tienen costes iniciales menores, proporcionalmente más resultados inmediatos y un ciclo de vida más corto.

## **Prediseño**

La fase de prediseño incluye lo siguiente: plan de diseño, flujos de procesos (físicos), estándares de diseño, prototipo conceptual de pantallas y evaluación del prediseño.

## **Plan de diseño**

El plan para la fase de diseño debe trasladarnos confortablemente desde el diseño lógico al diseño físico. El plan debe contener, por tanto, los resultados de la consulta a los usuarios sobre qué debe formar parte del nuevo sistema y qué debe permanecer como un proceso manual. El siguiente paso del plan debe ser un desarrollo iterativo de prototipos del sistema con diseños de pantalla que son soportados por flujos de proceso de nivel físico; cuando los desarrolladores están convencidos de tener un diseño robusto, deben volver a la jerarquía funcional y rehacerla a nivel físico. Lo siguiente sería identificar las funciones elementales como preparación para la generación de módulos.

## **Flujos de procesos (físicos)**

Los flujos de procesos físicos difieren de los lógicos en que los flujos lógicos modelan el negocio, mientras que los físicos modelan el sistema propuesto.

El desarrollo de flujos de procesos lógicos es el primer intento de un diseño funcional del sistema. Estos flujos de procesos describen la manera en que el nuevo sistema trabajará funcionalmente. Se irán modificando en la fase de diseño según vaya madurando el diseño del sistema.

## **Estándares de diseño**

Los estándares de diseño describen la composición del sistema propuesto. Se deben tomar decisiones en función de lo siguiente:

- Definición de pantallas, incluyendo colores, fuentes y botones.
- Herramientas de navegación, incluyendo menús y botones.
- Ayuda.
- Documentación.
- Funcionalidad.
- Estándares de programación.
- Convenciones de nombrado.



Una de las mejores estrategias para el desarrollo de estándares de diseño es crear unas cuantas aplicaciones arquetipo y efectuar una ingeniería inversa de los estándares a las plantillas

### **Prototipo conceptual de pantallas**

Se debe crear un prototipo de pantallas utilizando como base una gran parte del sistema. Este prototipo es una entrega fundamental que validará los estándares de diseño y análisis. Los prototipos de diseño de pantallas implementan los flujos de procesos físicos y los estándares de diseño. Los usuarios pueden evaluar, a través de los prototipos, la capacidad del sistema para satisfacer sus necesidades. No es necesario que el prototipo acceda a los datos, aunque cuando se está intentando que los usuarios comprendan la estrategia de diseño, un poco de funcionalidad siempre ayuda.

### **Evaluación del prediseño**

En la evaluación del prediseño, el protagonismo se desplaza desde el usuario hacia el sistema. En esta fase los profesionales de sistemas tienen más que decir que los usuarios. Los usuarios se tienen que sentir a gusto con el plan de diseño; pero el personal de sistemas será el que lo vaya a implementar, por lo que su influencia debería ser mayor que la de los usuarios.

La fase de prediseño se completa cuando los usuarios están satisfechos con los prototipos de pantalla y los flujos de procesos. Los estándares de diseño y el plan de diseño tienen que ser aprobados tanto por los usuarios como por los desarrolladores.

## **I.2.5 Diseño**

La fase de diseño es aquella en la que se dejan los anteproyectos para empezar a construir el sistema. Se deben afinar todos los detalles antes de empezar la generación. La fase de diseño se divide en dos partes: diseño de la base de datos y diseño de la aplicación.

### **Diseño de la base de datos**

Del diseño de la base de datos forman parte el diseño de las tablas y columnas junto con la especificación detallada de los dominios y verificación de las restricciones de las columnas. El diseño de la base de datos incluye también la desnormalización de la base de datos para mejorar el rendimiento junto con los disparadores asociados que soporten la desnormalización.

### **Diagrama E/R de diseño**

El diagrama entidad relación de diseño es el diagrama entidad relación final antes de la construcción. Ahora es cuando se debe realizar cualquier desnormalización necesaria. Hay que considerar los detalles de bajo nivel; y los diseños de las tablas deben ser rigurosos, probados lógicamente con datos de ejemplo antes de su implementación.

### **Diseño de aplicaciones**

Además del diseño de informes y aplicaciones específicas, el diseño de aplicaciones incluye la decisión sobre qué productos deben construirse.

### **Diseño de pantallas**

Debería construirse un prototipo del sistema completo basándose sólo en los diseños de pantallas. Lo que no hay que hacer es ir directamente a la generación de CASE sin haber desarrollado un prototipo. Los prototipos se pueden crear con CASE utilizando herramientas de desarrollo GUI.

## **Libro de diseño**

Cada módulo que vaya a ser generado tendrá un conjunto de informaciones asociadas a él. Además de la información asociada a la herramienta, cada módulo tiene funciones que deben ser documentadas. El objetivo sería generar, por cada aplicación, una carpeta que describa totalmente esa aplicación. Esta carpeta tendría el papel de unidad primaria del documento de pruebas. Además, los flujos de procesos físicos serían parte del libro del diseño para asistir las pruebas del sistema.

## **Evaluación del diseño**

El diseño estará completo cuando se puedan traspasar a otro equipo los documentos de diseño para la fase de construcción, teniendo cada aplicación su propio diseño de pantallas (o informes), lista detallada de funcionalidades e informe de creación-recuperación-actualización-eliminación.

### **I.2.6.- Construcción**

La fase de construcción comprende dos áreas: la base de datos y las aplicaciones. La construcción de la base de datos supone su generación directa mediante CASE. Todos los disparadores y estructuras de datos pueden mantenerse con el modelo físico del CASE.

El único trabajo específico que hay que efectuar en este punto es decidir sobre la cantidad de espacio reservado a las tablas y las localizaciones del disco físico. También hace falta tomar una decisión sobre cuántas instancias se van a utilizar. Una instancia es un entorno autocontenido. Lo normal es utilizar tres (desarrollo, pruebas y producción).

Antes de construir aplicaciones, se necesita una base de datos de ejemplo para probar si las aplicaciones trabajarán correctamente. Es posible y corriente construir sistemas sin esta base de datos de ejemplo. Sin embargo, en esos casos, se puede perder una gran cantidad de tiempo calculando por dónde vendrán los errores y cuán rápido funcionará el sistema con datos reales.

Siempre que sea posible deberían utilizarse plantillas en la construcción de las aplicaciones. Después de la generación, la modificación de aplicaciones debe almacenarse en plantillas específicas de cada aplicación para que sirvan de ayuda en la regeneración. Este esfuerzo minimiza la necesidad de modificaciones adicionales tras la regeneración.

Antes de finalizar la fase de construcción, se debe someter al sistema a una primera pasada de pruebas a nivel de unidad. Hay que asegurarse de realizar esta tarea mientras se construye la aplicación, porque todavía se tiene la aplicación fresca en la mente. E inmediatamente después pasar el sistema a la persona que va a realizar las pruebas para comprobar que el sistema satisface las especificaciones del libro de diseño y que funcionalmente trabaja correctamente desde el punto de vista del usuario.

### **I.2.7.- Implantación**

#### **Documentación**

La documentación debiera ser un proceso continuo a lo largo de todo el proceso de desarrollo del sistema. Debe acompañar el primer prototipo que el usuario vea. La documentación no debe ser simplemente un paso separado al final del proceso.

Así como las aplicaciones del sistema se prueban, la documentación también debe pasar un proceso de pruebas. Probar el sistema en sí es imposible antes de escribir la documentación del sistema, puesto que no hay nada contra lo que comprobar. El asunto es que ningún sistema está acabado hasta que tanto la documentación del sistema como la de usuario estén finalizadas satisfactoriamente.

## Pruebas

Las pruebas son una de las más importantes pero normalmente peor realizadas fases en el proceso de diseño de sistemas. La clave para probar correctamente es utilizar pruebas múltiples. Ninguna prueba única, no importa cuán cuidadosamente efectuada, encontrará todos los errores del sistema. Es mejor realizar unas cuantas pruebas diferentes con menos cuidado; normalmente esto descubre más errores a menor coste para la organización.

Cuando se alcanza la fase de pruebas del proceso de desarrollo no se necesita revisar el diseño lógico o físico de la base de datos. Esto ya se realizó al final de las fases de análisis y diseño. El primer objetivo de esta etapa es revisar la corrección de las aplicaciones.

Para conseguir este objetivo, se deben realizar pruebas a nivel de unidad aplicación por aplicación. El proceso de comprobación debe ser meticuloso, utilizando datos de prueba, programas de prueba automatizados, inspecciones de código y entrevistas con el desarrollador del sistema. También deberán realizarse verificaciones globales a nivel de base de datos, incluyendo los siguientes procesos:

- Ejecutar todas las aplicaciones con una plataforma (gran cantidad de datos de ejemplo) de prueba.
- Verificar la consistencia de la base de datos ejecutando muchos procedimientos.
- Comparar la salida de los informes del nuevo sistema con la del antiguo.
- Construir pequeñas aplicaciones de formularios simples para ayudar en la inspección de los datos.
- Ejecutar filtros en todas las columnas de todas las tablas para verificar valores válidos de los datos.

El segundo objetivo de la fase de pruebas es realizar una prueba de aceptación del usuario. Proporcionar a los usuarios las aplicaciones para que trabajen con ellas y realicen transacciones reales (no de ejemplo) utilizando el nuevo sistema.

## Implementación

En algún momento, el sistema finalizado tiene que ser traspasado a los usuarios y puesto en marcha (después de que haya tenido lugar, por supuesto, la formación al usuario sobre el nuevo sistema). Hay varias escuelas de pensamiento sobre la implementación. La primera es la aproximación “big bang”. La segunda es la implementación por fases.

La aproximación big bang implica desenchufar el sistema antiguo, levantar el nuevo e insistir en que todo el mundo lo use. Con esta aproximación, no hay vuelta atrás. Si se usa esta aproximación, es crucial que el sistema haya sido bien probado.

La ventaja de esta aproximación es que es más barato hacer todo de una vez y no tener sistemas paralelos ejecutándose simultáneamente. En teoría, esta aproximación implica mucho menos trabajo porque fuerza el ciento por ciento de compromiso de toda la organización.

La mayor desventaja de la aproximación big bang es que no hay ninguna manera de probar un sistema tan extensamente como para que un desarrollador pueda garantizar que funcionará. Sin embargo, si realmente tienes confianza en el sistema, esta aproximación puede ahorrar tiempo y dinero; tanto como se esté dispuesto a asumir los riesgos.

La segunda aproximación, más común, es la implementación por fases. Varias partes del nuevo sistema se levantan al mismo tiempo, bien por ternas o bien por clases de usuarios. En la mayoría de los casos, ejecutar el nuevo sistema en paralelo con el heredado requerirá doble entrada en ambos sistemas mientras se depuran los errores. Como aproximación es claramente más costosa

que la aproximación big bang, pero si algo va mal, siempre se tiene el sistema heredado para recurrir a él.

Otras partes importantes de la fase de implementación son idear una planificación de la implementación y una estrategia de formación del usuario. ¿Cómo se formará a un sistema de aprendizaje basado en ordenador (CBT, Computer Based Training). Sin embargo, excepto para las organizaciones más grandes, la creación de un sistema de aprendizaje basado en ordenador es muy lenta y demasiado costosa como para que sea efectiva. Como el sistema evoluciona y cambia, la modificación del sistema de aprendizaje basado en ordenador es cara y consume mucho tiempo.

Otra consideración en la fase de implementación es la estrategia de ayuda al usuario (help desk). ¿Cómo dar soporte a los usuarios del nuevo sistema? La estrategia de ayuda al usuario puede incluir:

- Una sola persona que esté formada para dar soporte.
- Unos cuantos expertos en partes el nuevo sistema que reciben formación extra.
- Manuales de usuario
- Correo electrónico o soporte telefónico.

La estrategia de ayuda al usuario debe ser cuidadosamente considerada antes de que entre en funcionamiento el nuevo sistema.

### **I.2.8.- Mantenimiento**

Incluso cuando un sistema está “finalizado” y puesto en marcha, está cambiando continuamente. Habrá, por supuesto, algunos problemas con cualquier nuevo sistema junto con la necesidad de aumentar los usuarios, peticiones de cambios de funcionamiento del sistema, la necesidad de nuevos informes, campos desaparecidos, etcétera. El objetivo principal de la fase de mantenimiento es proporcionar un proceso que pase por un filtro, clasifique y después gestione estos problemas y cambios del sistema. Es necesario reconocer estos cambios, que no sólo afectan a la base de datos y a las aplicaciones, sino que también deben ser reflejados en la documentación del usuario y del sistema, y en la formación. Al personal involucrado en las funciones de ayuda al usuario se le debe mantener informado de cualquier cambio.

La gestión de versiones es la clave de un mantenimiento de sistemas eficiente. No se puede ir haciendo cambios al sistema uno por uno e ir implementándolos por todo el sistema.

Estos cambios incontrolados son potencialmente peligrosos para la integridad del sistema de producción. Los cambios deben ser efectuados acompañados de un proceso que incluya pruebas y garantía de calidad (QA, Quality Assurance).

Finalmente, los cambios no son baratos de hacer. Para minimizar costes, los cambios deseados se deben aunar con toda la documentación asociada y con actualizaciones de la ayuda al usuario. Debido a que, frecuentemente, los proyectos pueden tardar meses e incluso años en completarse, es probable que cuando se alcance la fase de implementación, los requisitos del sistema sufran cambios respecto a los determinados en la fase de análisis.

## REFERENCIAS BIBLIOGRÁFICAS

- [1SE02] +1 Software Engineering. 2002. 2510-G Las Posas Road, Suite 438, Camarillo, California 93010, <http://www.cs.queensu.ca/softwareengineering/tools.html>.
- [ALL87] Allen, J. 1987. Natural Language Understanding. Menlo Park, CA: Benjamin/Cummings.
- [APV02] Artemis Project View. 2002. <http://www.artemisp.com>.
- [BOE00] Bohem, 2000, Software Cost Estimation with COCOMO II, Barry Bohem, Chris Abats, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horwitz, Ray Madachy, Donald J. Reifer and Bert Steece. Ed Prentice Hall PTR.
- [BOE88] Boehm, B., "A Spiral Model for Software Development and Enhancement", Computer, vol. 21, no. 5, mayo 1988, pp. 61-72.
- [BOE98] Boehm, B., "Using the WINWIN Spiral Model: A Case Study", Computer, vol.31, no. 7, julio 1998, pp. 33-44.
- [BRO75] Brooks, F., The Mythical Ma-Month, Addison- Wesley, 1975.
- [BUR76] Burton, R. R. 1976. Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems. Tech. Rep. 3453, Bolt Beranek and Newman, Boston, MA.
- [CAI98] Osvaldo Cairo. 1998. KAMET: A comprehensive methodology for knowledge acquisition from multiple knowledge sources. Expert Systems with Applications, An International Journal Volume 14, Number 1 / 2, January/February.
- [CLI97] Joseph C. Giarratano, Ph. D. 1997. CLIPS User's Guide. <http://www.jsc.nasa.gov/clips/CLIPS.html>.
- [CMI84] Clocksin, W. F. y C. S. Mellish. 1984. Programming in Prolog, 2d ed. Nueva York. Springer-Verlag.
- [CRR02] Rational Rose. 2002. <http://www.rational.com/products/rose>.
- [CUL86] Cullingford, R. E. 1986. Natural Language Processing. A Knowledge Engineering Approach. Totowa, NJ: Rowman y Allanheld.
- [CWA96] Comyn-Wattiau I., Akoka, J. 1996.: Reverse Engineering of Relational Database Physical Schemas., Proceedings 15th International Entity Relationship Conference, Cottbus, Allemanne.

## REFERENCIAS BIBLIOGRÁFICAS (Continúa)

- [CH96a] Chen, H. Machida, K., Far, B. H., Koono, Z. 1996 : Software Creation: A Systematic Construction Method of Expert Systems Used for Design, 3th World Congress on Expert System, Seoul, Korea, pp 577-584.
- [CHM98] Hui Chen, Nagayasu Tsutsumi, Zenya Koono. 1998.: Software Creation: An Intelligent CASE Tool for Reusing Design Knowledge Acquired Automatically. 4th World Congress on Expert Systems. Application of Advanced Information Technologies. Vol 1. Ed.: Cantu, Soto, Liebowitz, Sucar: México D.F.
- [CHN97] Chen, H; Far, B.H;Koono, Z. 1987; A Systematic Construction Method of an Expert System Used for Automatic Design – Acquisition and Reproduction of Design Knowledge from Design Process, Journal of Japan Society for Artificial Intelligence, Vol. 12 No.4. pp. 616-626.
- [CHO57] Chomsky, N. 1957. Syntactic Structures. La Haya: Mouton.
- [CHS98] Pavol Návrát, Haruki Ueno. 1998. Knowledge-Based Software Engineering. Conference: Software Creation: An Intelligent CASE Tool for Reusing Elementary Design Knowledge. Authors: Hui Chen, Nagayasu Tsutsumi, Zenya Koono: Slovakia.
- [DES00] Designer 2000, Manual de Oracle Designer 2000, Paul Dorsey, Peter Koletzke, Versión 1.3 , Oracle Press, McGraw Hill.
- [DOW85] Dowty, D. R., L. Karttunen, y A.M. Zwinky. 1985. Natural Language Parsing: Psychological, Computacional and Theoretical Perspectives. Nueva York: Cambridge University Press.
- [EDY93] Edward Yourdon. 1993. “Análisis estructurado moderno”. Ed. Prentice Hall.
- [FBD93] Korth Silberschatz. 1993. “Fundamentos de base de datos”. Ed. Mc-Graw-Hill.
- [GRO86] Grosz, B. J., K. Spark Jones, y B. L. Webber. 1986. Readings in Natural Language Processing. Los Altos, CA: Morgan Kaufmann.
- [HEL81] Hendrix, G. G. y W. H. Lewis . 1981. Transportable natural-language interfaces to databases. En Proceedings of the 19th Annual Meeting of Association for Computacional Linguistics.
- [HEN84] Henderson P. (ed.), Software Engineering Symposium on Practical Software Development Enviroments, ACM Sigsoft Notes, vol. 9, no. 3, May 1984.
- [IAS02] Ian Sommerville, sexta edición 2002, editorial Addison Wesley.

## REFERENCIAS BIBLIOGRÁFICAS (Continúa)

- [JAA98] Jacky Akoka. 1998. Merci: An Expert System for Software Reverse Engineering. 4th World Congress on Expert Systems. Application of Advanced Information Technologies. Vol 1. Ed.: Cantu, Soto, Liebowitz, Sucar: México D.F.
- [JES02] JESS, 2002 Java Expert System Shell <http://herzberg.ca.sandia.gov/jess/>, Friedman.
- [JOS81] Joshi, A. K., B. L. Webber, y I. A. Sag, eds. 1981. Elements of Discourse understanding. Cambridge: Cambridge University Press.
- [KEK94] Keving Knigth, 1994, y Elaine Rich, Inteligencia Artificial, Segunda edición McGraw-Hill.
- [KIN83] King, M. 1983. Parsing Natural Language. Nueva York: Academic Press.
- [LEB88] Lehrberger, J. L. Bourbeau. 1988. Machine Translation: Linguistic Characteristics of MT Systems and General Methodology of Evaluation. Philadelphia: Benjamins.
- [MCB88] McDonald, D. D. y L. Bolc. 1988. Natural Language Generation Systems. Nueva York: Sringer-Verlag.
- [MCS87] McKeown, K. R. y W. R. Swartout. 1987. Languaje generation and explanation. En Annual Review of Computer Science. Volume 2. Palo Alto: Annual Reviews.
- [MIN75] Minsky, M. 1975. A framework for representing knowledge. En The Psycology of Computer Vision, ed. P. Winston. Nueva York: MacGraw-Hill.
- [MOD83] Moder, J. J., C. R. Philips y E. W. Davis, 1983. "Project Management with CPM, PERT and Precedence Diagramming", 3 ed., VanNostrad Reinhold
- [MSP02] Microsoft Project. 2002.  
[http://www.stylusinc.net/msproject\\_tutorial/project\\_management.shtml](http://www.stylusinc.net/msproject_tutorial/project_management.shtml).
- [NAG89] Nagao, M. 1989. Machine Translation Summit. Tokyo: Ohmsha.
- [NAU69] Naur, P., y B. Randell (eds.), Software Engineering: A Report on a Conference sponcered by the NATO Science Comittee, NATO, 1969.
- [NEW82] Newell, A. 1982. The Knowledge level. Artificial Intelligence 18(1).
- [NIL80] Nilsson, N. J. 1980. Principles of Artificial Inteligence. Palo Alto, CA: Morgan Kaufmann.

## REFERENCIAS BIBLIOGRÁFICAS (Continúa)

- [NIÑO00] Olivia Niño, L. 2000. Herramientas para estimar factores de riesgo en Proyectos de Software. Instituto Mexicano del Petróleo. Nino@imp.mx: México D.F.
- [NIR87] Nirenburg, S. 1987. Machine Translation: Theoretical and Methodological Issues. Cambridge, England: Cambridge University Press.
- [PEN02] Primavera Enterprise. 2002. <http://www.primavera.com>.
- [PMI96] William R. Duncan. 1996. PMI Standards Committee, "A guide to the Project Management Body of Knowledge".
- [POO98] Luis Joyanes Aguilar. 1998. Programación Orientada a objetos, segunda edición. Ed. Osborne Mac-Graw-Hill.
- [PSP99] SAP R/3 segunda edición, 1999, Edición especial, Prentice Hall.
- [PUT78] Putnam, L. 1978. "A General Empirical Solution to the Macro Software Sizing and the Estimation Problem", IEEE Trans. Software. Engineering, vol. 4, no. 4. pp. 345-361.
- [PUT92] Putnam, L., y W. Myers, 1992. "Measures for Excellence", Yourdon Press.
- [QUL78] Quine, W. V. y J. S. Ullian. 1978. The Web of Belief. Nueva York: Random House.
- [RPE02] Roger S. Pressman, 2002. Ingeniería del software, un enfoque práctico. Quinta Edición, Ed. Mc-Graw-Hill.
- [RPE90] Roger S. Pressman, 1990. Ingeniería del software, un enfoque práctico. Segunda Edición, Ed. Mc-Graw-Hill.
- [SAM63] Samuel, A. L. 1963 Some studies in machine learning using the game of checkers. En Computer Models of Thought ed. E. A. Feigenbaum y J. Feldman. Nueva York:McGraw-Hill.
- [SCP93] Sergio V. Chapa V., Pedro Alday E. 1993, Perspectivas en la Automatización de las Metodologías de Ingeniería de Software. CINVESTAV del IPN: México D.F.
- [SHA73] Schank, R. C. 1973. Identification of conceptualizations underlying natural lenguaje. En Computer Models of Thought and Lenguaje. ed. R. C. Schank y K. M. Colby. San Francisco: Freeman.
- [SHA75] [SHA75] Schank, R. C. 1975. Conceptual Information Processing. Amsterdam: North-Holland.



## REFERENCIAS BIBLIOGRÁFICAS (Continúa)

- [SHA77] [SHA77] Shank, R. C. y R. P. Abelson. 1977. Scripts, Plans, Goal, and Understanding. Hillsdale, NJ: Erlbaum.
- [SLO88] [SLO88] Slocum, J. 1988. Machine Translation Systems. Cambridge: Cambridge University Press.
- [SRU96] [SRU96] Stuart Russell, Peter Norving, 1996. Inteligencia Artificial, un enfoque moderno. Ed. Prentice Hall.
- [TPW02] [TPW02] Thayer, Pyster y Wood. 2002.  
<http://www.itcg.edu.mx/ingsoft/admsoft.htm>
- [W&C94] [W&C94] Waters, R. C., Chikofsky, E. 1994. Reverse Engineering – Progress Along Many Dimensions. Communications of the ACM., Vol 37(5).
- [WIL72] [WIL72] Wilks, Y. A. 1972. Grammar, Meaning and the Machine Analysis of Language. Londres: Routledge and Kegan Paul.
- [WIN83] [WIN83] Winograd, T. 1983. Language as a Cognitive Process: Syntax. Reading, MA: Addison-Wesley.

## GLOSARIO DE TERMINOS

CADM.- Son las siglas en inglés de *CASE Application Development Method*, en español significa Método de Desarrollo de Aplicaciones CASE, es una metodología que se utiliza como herramienta para apoyar el desarrollo de *software*.

CASE.- Son las siglas en inglés de *Computer Aided Software Engineering*, en español Ingeniería de *Software* Asistida por Computadora, comprende un amplio rango de herramientas que se utilizan para ayudar a las actividades del desarrollo de *software*, como el análisis de requerimientos, el modelado de sistemas, la depuración y las pruebas.

Clase.- Es una forma de representar una plantilla para varios objetos con características similares, por ejemplo la clase árbol que describe las características de todos los árboles tiene hojas, raíces, crecen, producen clorofila. Entonces la clase árbol sirve como un modelo abstracto para el concepto de un árbol.

CLIPS.- Son las siglas en inglés de *C Language Integrated Production System*, es un lenguaje que sirve para programar reglas de producción de los sistemas expertos, y que se procesan en un mecanismo de inferencia para deducir nuevo conocimiento a partir de un conocimiento inicial existente.

COCOMO.- Es un modelo empírico que se obtuvo recolectando datos de varios proyectos de *software* grandes y después analizando los datos para descubrir fórmulas que se ajustarán mejor a las observaciones. Se han realizado varias versiones de este modelo entre ellos el COCOMO 81 y el último es COCOMO II, que considera diferentes enfoques de desarrollo de *software*, desde prototipos hasta métodos más complejos.

Dependencias Conceptuales.- Con frecuencia se representa en inglés como CD, y es una teoría sobre la representación del tipo de conocimientos sobre eventos que normalmente aparecen en las frases de lenguaje natural.

FCA.- Son las siglas de Fuentes de Conocimiento Activo, está formado por el grupo de expertos humanos que aportaran su conocimiento, experiencia y destreza dentro de una especialización determinada de una empresa, los expertos humanos se pueden organizar de diversas formas para aportar su conocimiento para el desarrollo de una aplicación con técnicas de inteligencia artificial.

FCP.- Son las siglas de Fuentes de Conocimiento Pasivo, está formado por manuales, catálogos, libros de texto o cualquier otro material escrito, asociado al dominio del conocimiento de una aplicación a desarrollar.

ICASE.- Significa Intelligence CASE, y utiliza técnicas de la inteligencia artificial asociadas a las capacidades de las herramientas CASE.

Ingeniería de *Software*.- Existen varias definiciones de este concepto, dentro de estas algunas de las definiciones son: a) "Es una disciplina o área de la Informática o Ciencias de la Computación que ofrece métodos y técnicas para desarrollar y mantener *software* de

calidad que resuelven de todo tipo de proceso”, b) “Es el establecimiento de los principios y métodos de la Ingeniería a fin de obtener *software* de modo rentable que sea fiable y trabaje en máquinas reales”.

Ingeniería inversa.- Es el proceso de analizar el *software* con el objetivo de recuperar su diseño y especificación. Por lo regular, el código fuente de *software* es la entrada al proceso de ingeniería inversa.

Instancia.- Una instancia pertenece a una clase y a diferencia de ésta, la instancia es la representación concreta de un objeto y contiene los atributos específicos del objeto por ejemplo las ramas del árbol son cafés, las hojas tienen 3 centímetros de largo, etc.

Inteligencia Artificial.- Existe varias definiciones, una de ellas es que estudia como lograr que las máquinas computadoras realicen tareas que, por el momento, son realizadas mejor por los seres humanos.

KAMET.- Son las siglas en inglés de *Knowledge Acquisition from Multiple Sources*, y es una metodología que apoya el desarrollo de sistemas expertos basados en conocimiento, esta orientada a resultados.

Lenguaje Natural.- Se le denomina a las diferentes formas de comunicarse entre los seres humanos; oral y escrito, a la manera de procesar este lenguaje en un sistema inteligente se le llama procesamiento de lenguaje natural e involucra varias etapas que interpretan fundamentalmente las formas gramaticales.

Marcos.- Se le define como una colección de atributos con valores asociados y posibles restricciones entre valores, que describe de forma abstracta alguna entidad u objeto del mundo real. Esta definición tiene una gran similitud con la definición de clases de objetos.

Metodología.- Es un conjunto de etapas y actividades que conforman el proceso de desarrollo del *software*, en algunos casos también se les puede denominar paradigmas de *software* que se aplican de acuerdo a la situación y características del sistema o *software* a desarrollar.

Métricas de *software*.- Son mediciones que se aplican en el desarrollo del *software* por computadora. Estas mediciones se pueden aplicar a los procesos o metodologías de *software* con el intento de mejorarlos sobre una base continua de registros de indicadores y estadísticas.

PMI.- Son las siglas en inglés de Project Management Institute, y es una institución a nivel internacional que emite y certifica mejores prácticas para la administración de proyectos.

Procedimiento.- Es una secuencia de pasos ordenados que describen en un flujo lógico las actividades por realizar en un proceso.

Prototipo.- Un prototipo es la construcción de un primer sistema que sirve para identificar los requisitos o el dominio de un sistema o *software* a desarrollar.

Proyecto.- Es un compromiso temporal para crear un producto o servicio único. Temporal

significa que cualquier proyecto tiene un inicio y un fin definidos. Único significa que el producto es diferente en algo que lo distingue de otros productos o servicios similares.

Puntos de Función.- Es un conjunto de métricas de *software* que se derivan con una relación empírica según las medidas contables del dominio del *software* a desarrollar, entre ellas: número de entradas del usuario, número de salidas, peticiones, archivos y número de interfaces.

Redes de actividades.- Es un conjunto de actividades que están ligadas entre sí con un orden de precedencias, que permite establecer una secuencia exacta de tiempo entre ellas.

Ruta crítica.- Es la ruta más larga entre un punto en la red de actividades, con un arco representando una actividad que no tiene predecesores y el extremo de otro arco que representa una actividad que no tiene sucesores. La razón del nombre se debe a que la longitud (tiempo) de esta trayectoria es el mínimo tiempo en el cual se puede terminar el proyecto; cualquier retraso en una actividad en esta ruta alargará necesariamente todo el proyecto.

SE-APIS.- Son las siglas en español de Sistema Experto como apoyo a la Administración de Proyectos de Ingeniería de *Software*, y es el sistema desarrollado en este trabajo de tesis.

HIS.- Son las siglas en español de Herramienta de Ingeniería de *Software*, y que sirve de base conceptual para el desarrollo del prototipo del SE-APIS.

Sistema Experto.- Es un sistema desarrollado con técnicas y tecnologías de la Inteligencia Artificial y que persigue procesar una área especializada del conocimiento humano, para sustituir o apoyar al experto humano en la realización de sus actividades. Actualmente puede recibir también los nombres de sistema basado en conocimiento, sistema inteligente o agente inteligente.

WBS.- Son las siglas en inglés de *Work Breakdown Structure*. Un WBS está orientado a entregables agrupados de elementos de un proyecto, que organizan y definen el total del alcance del proyecto, el trabajo que no está en el WBS no está en el proyecto. El WBS es frecuentemente usado para desarrollar y confirmar un conocimiento común del alcance del proyecto. Está organizado en niveles jerárquicos en donde cada nivel descendente representa un incremento en el detalle de la descripción de los elementos del proyecto.