

UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POSTGRADO



**METRICAS DE HALSTEAD EN LENGUAJES
MANIPULADORES DE BASES DE DATOS**

TESIS
EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE
LA ADMINISTRACION CON ESPECIALIDAD
EN SISTEMAS

POR
ING. LUIS GERARDO NAVARRO GUERRA

SAN NICOLAS DE LOS GARZA, N. L. ENERO DE 1997

METRICAS DE HALSTEAD EN LENGUAJES
MANIPULADORES DE BASES DE DATOS

L.G.N.G.

FM
Z5853
.M2
FIME
1997
N3

I

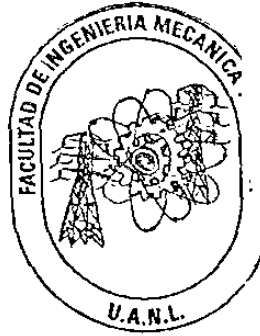


1020119021

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POSTGRADO



**METRICAS DE HALSTEAD EN LENGUAJES
MANIPULADORES DE BASES DE DATOS**

T E S I S

**EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA
ADMINISTRACION CON ESPECIALIDAD EN SISTEMAS**

P O R

ING. LUIS GERARDO NAVARRO GUERRA

SAN NICOLAS DE LOS GARZA, N.L.

ENERO DE 1997

TM
25853
.M2
FINE
1997
N3

0119-58760




FONDO TESIS

**UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION DE ESTUDIOS DE POSTGRADO**

Los miembros del comité de tesis recomendamos que la tesis METRICAS DE HALSTEAD EN LENGUAJES MANIPULADORES DE BASES DE DATOS realizada por el Ing. Luis Gerardo Navarro Guerra sea aceptada para su defensa como opción al grado de Maestro en Ciencias de la Administración con especialidad en Sistemas.

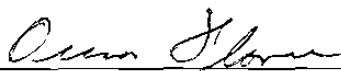
El Comité de Tesis




Asesor
Dr. José Luis Martínez Flores



Coasesor
Dra. Ada Margarita Álvarez Socarrás



Coasesor
Dr. Oscar Flores Rosales



Vo. Bo.
M.C. Roberto Villarreal Garza
División de Estudios de Postgrado

San Nicolás de los Garza, N. L. a 15 de Enero de 1997

DEDICATORIAS

A mis padres: *Julio Navarro Mercado* y *María Ana Guerra de Navarro*, por estar en todo momento apoyándome incondicionalmente.

A mis hermanos: *Ana Lilia*, *Julio César*, *Juan José* y *Victor Hugo*. A mi cuñada *Sylvia M. G.* y a mi sobrina *Sylvia Carolina*, por su apoyo y comprensión brindados en todas las etapas de mi vida.

A mi novia: *Norma Edith* por el cariño, comprensión y apoyo desinteresado durante todo el tiempo que llevamos juntos.

A todas las personas que de alguna forma han estado conmigo brindándome su amistad, apoyo y comprensión en todo momento de mi existencia.

AGRADECIMIENTOS

Deseo expresar mi más sincero agradecimiento al Dr. José Luis Martínez Flores, asesor de esta tesis, por su valiosa ayuda y su acertada guía para el desarrollo del presente trabajo.

También deseo agradecer, a los coasesores, Dra. Ada Margarita Alvarez Socarrás y Dr. Oscar Flores Rosales, todos sus invaluable consejos y recomendaciones para concluir esta tesis.

En forma especial, agradezco a todos mis compañeros del Programa Doctoral en Ingeniería de Sistemas (DIS) por su amistad, compañía y apoyo brindado durante el desarrollo de la tesis.

Por último, agradezco a la Universidad Autónoma de Nuevo León y al Consejo Nacional de Ciencia y Tecnología (CONACYT) toda la ayuda proporcionada durante mi etapa de postgrado.

RESUMEN

Luis Gerardo Navarro Guerra

Fecha de Graduación: Enero, 1997

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Título del Estudio: **MÉTRICAS DE HALSTEAD EN LENGUAJES
MANIPULADORES DE BASES DE DATOS**

Número de Páginas: 92

Candidato para el grado de Maestría
en Ciencias de la Administración con
especialidad en Sistemas

Area de Estudio: Métricas de Software

Propósito y Método del Estudio: El propósito principal de este trabajo es determinar si las Métricas de Halstead son aplicables a lenguajes manipuladores de bases de datos (LMBD). Las preguntas de investigación son: 1) ¿Es la métrica de longitud del programa propuesta por Halstead un buen estimador del tamaño de los programas escritos en un LMBD? 2) ¿Afecta el tamaño y la función de un programa en la estimación del tamaño de programas escritos en los LMBD? 3) ¿Es el nivel de lenguaje de los LMBD mayor que los lenguajes de tercera generación (3GL's) y menor que el Inglés Prosaico? Se realizó un estudio ex-post-facto, utilizando un analizador de código como instrumento para la recolección de datos. Los LMBD seleccionados para este estudio fueron PROGRESS y ORACLE, siendo la muestra de 957 y 63 programas de código fuente, respectivamente. Para responder a las preguntas de investigación se realizaron análisis de correlación y pruebas de hipótesis (pruebas Z).

Contribuciones y Conclusiones: Para el lenguaje PROGRESS se encontró que: a) las métricas de longitud y de nivel de lenguaje propuestas por Halstead son válidas; b) la estimación de un programa depende del tamaño y de la función que realiza. Para el lenguaje ORACLE sólo se encontró que la métrica de longitud de programa fue válida. Los resultados de este estudio pueden utilizarse para medir el desempeño de un grupo de programadores, establecer niveles del lenguaje en el desarrollo de programas, evaluar la calidad de programas existentes y planear nuevos proyectos.

FIRMA DEL ASESOR:

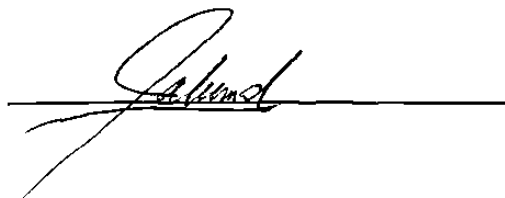


TABLA DE CONTENIDO

Capítulo	Página
1. INTRODUCCION	1
1.1. Establecimiento del Problema.....	1
1.2. Objetivo del Estudio	2
1.3. Justificación del Estudio	3
1.4. Limitaciones del Estudio	4
2. ANTECEDENTES	5
2.1. Introducción	5
2.2. Software	6
2.2.1. Concepto de Software	6
2.2.2. Clasificación del Software.....	7
2.2.3. Lenguajes de Programación	8
2.2.4. Lenguajes Manipuladores de Bases de Datos.	12
2.3. Ingeniería del Software	13
2.3.1. Elementos Clave de la Ingeniería del Software.....	13
2.3.2. Paradigmas de la Ingeniería del Software	15
2.3.2.1 El Ciclo de Vida Clásico.....	15
2.3.2.2 Construcción de Protótipos.....	17
2.3.2.3 Modelo en Espiral.....	19
2.3.3. Unificación de los Paradigmas de la Ingeniería del Software.....	21
2.3.3.1. Fase de Definición	22
2.3.3.2. Fase de Desarrollo.....	22
2.3.3.3. Fase de Mantenimiento	23
2.3.4. Problemas con el Desarrollo de Software	24
2.3.5. Problemas Relacionados a las Metodologías Actuales	25
2.3.5.1. Inflexibilidad.....	25
2.3.5.2. Mantenimiento	25

2.3.5.3. Acumulación en el Desarrollo de Aplicaciones	26
2.4. Métricas de Software	27
2.4.1. Medición y Métricas de Software	27
2.4.2. Clasificación de Métricas de Software	28
2.4.3. Implementación de Métricas de Software	29
2.4.3.1. Enfoque para la Implementación de Métricas de Software .	30
2.4.3.2. Problemas en la Implementación de Métricas de Software .	31
2.5. Métricas de Calidad del Software	32
2.5.1. Calidad del Software	32
2.5.2. Métricas Cualitativas	34
2.5.3. Métricas Cuantitativas	36
2.6. Resumen	37
3. METRICAS DE HALSTEAD	38
3.1. Introducción	38
3.2. La Ciencia del Software	38
3.2.1. Críticas a la Ciencia del Software	39
3.2.2. Aplicaciones de la Ciencia del Software	39
3.3. Métricas Básicas	40
3.4. Estimadores de un Programa	42
3.4.1. Estimador de la Longitud	42
3.4.2. Estimador del Volumen Potencial	44
3.4.3. Estimador del Nivel de Programa	44
3.4.4. Estimador del Esfuerzo y Tiempo de Programación	45
3.5. Nivel del lenguaje de Programación	47
3.6. Resumen	50
4. METODOLOGIA DE LA INVESTIGACION	51
4.1. Introducción	51
4.2. Preguntas e Hipótesis de la Investigación	51
4.3. Diseño de la Investigación	53
4.4. Selección de los LMBD	53
4.5. Selección de la Muestra	54
4.6. Instrumento de Medición	54
4.7. Resumen	55

Capítulo	Página
5. ANALISIS DE DATOS.....	56
5.1. Introducción	56
5.2. Exactitud de los Datos	56
5.3. Estadísticas Descriptivas.....	58
5.4. Análisis de Datos para las Preguntas de la Investigación.....	72
5.5. Resumen.....	77
6. CONCLUSIONES	78
6.1. Introducción	78
6.2. Objetivo del Estudio	78
6.3. Discusiones y Conclusiones.....	79
6.3.1. Discusión y Conclusión de la Primer Pregunta de Investigación.....	79
6.3.2. Discusión y Conclusión de la Segunda Pregunta de Investigación... ..	80
6.3.3. Discusión y Conclusión de la Tercer Pregunta de Investigación.	81
6.4. Recomendaciones	83
REFERENCIAS.....	85
APENDICE -ANALIZADOR DE CODIGO.....	88

LISTA DE TABLAS

Tabla	Página
I. Medias y Varianzas del Nivel del Lenguaje para Cinco Lenguajes.....	48
II. Distribuciones de Frecuencia de Niveles del Lenguaje	48
III. Medias y Varianzas del Nivel del Lenguaje para Lenguajes de Cuarta Generación	49
IV. Distribuciones de Frecuencia de Niveles del Lenguaje para Lenguajes de Cuarta Generación.....	49
V. Longitudes Reales y Longitudes Estimadas para ORACLE7.....	58
VI. Longitudes Reales y Longitudes Estimadas para PROGRESS6	59
VII. Nivel del Lenguaje para los Programas de ORACLE7.....	64
VIII. Nivel del Lenguaje para los Programas de PROGRESS6	65
IX. Nivel del Lenguaje Promedio para PROGRESS Y ORACLE	68
X. Distribuciones de Frecuencia de Niveles del Lenguaje para ORACLE7	68
XI. Distribuciones de Frecuencia de Niveles del Lenguaje para PROGRESS6	69
XII. Nivel del Lenguaje para Programas de PROGRESS6 Clasificados por Función y Tamaño	71

Tabla	Página
XIII. Desviación Estándar para Programas de PROGRESS6 Clasificados por Función y Tamaño	71
XIV. Coeficiente de Correlación de Pearson entre Longitud Real y Longitud Estimada.....	72
XV. Coeficiente de Correlación de Pearson entre Longitud Real y Longitud Estimada para Programas de PROGRESS6 Clasificados por Tamaño y Función	73
XVI. Prueba Z entre LMBD y 3GL's	75
XVII. Prueba Z entre LMBD y el Inglés Prosaico	76

LISTA DE FIGURAS

Figura	Página
1. Curva de Fallas del Software (Idealizada).....	7
2. Las Cuatro Generaciones de los Lenguajes de Programación.....	8
3. El Ciclo de Vida Clásico.....	16
4. Creación de Prototipos.....	18
5. El Modelo en Espiral.....	20
6. Mantenimiento del Software.....	26
7. Clasificación de Métricas de Software.....	29
8. Factores de Calidad del Software de McCall.....	33
9. Relación Ideal entre Longitud Real y Longitud Estimada, y Correlación Lineal Práctica.....	43
10. Relación Ideal entre λ y N.....	50
11. Diagrama de Flujo de Datos del Analizador de Código (Nivel 0).....	91
12. Diagrama de Flujo de Datos del Analizador de Código (Nivel 1).....	91

CAPITULO 1

INTRODUCCION

1.1. Establecimiento del Problema

En las últimas décadas la industria informática ha crecido y evolucionado rápidamente. En México, durante la década de los 90's se ha puesto mayor atención a las siguientes especialidades en el área de informática [INEGI, 1992]:

- Bases de Datos y Sistemas de Información.
- Computación Teórica.
- Informática Educativa.
- Ingeniería del Software.
- Inteligencia Artificial.
- Lenguajes de Programación.

Así mismo se ha incrementado la producción de software en los últimos años, principalmente en el nivel de aplicaciones. Por esta razón, existe en el mercado gran cantidad de programas de computadora, por lo cual surge la necesidad de establecer formas de medir la calidad del software que será utilizado en cualquier empresa, industria, comercio, escuela o centro de investigación.

La medición del software es importante ya que permite a los administradores y desarrolladores entender mejor el proceso de desarrollo así como el software que se

produce. Sin embargo, al tratar de medir el software surge el problema de qué vamos a medir y cómo lo vamos a medir.

Existen métricas que nos permiten medir la calidad del software en forma cualitativa o cuantitativa. Dentro de las métricas cuantitativas se encuentra la Ciencia del Software [Halstead, 1977], la cual permite evaluar las características del software en forma objetiva o directa.

Aunque las Métricas de Halstead han sido criticadas en sus fundamentos teóricos [Shen *et al*, 1983; Courtney *et al*, 1993] se han utilizado para hacer estimaciones y evaluaciones de diferentes tipos de programas, así como herramienta de entrenamiento para el desarrollo de software [Henry *et al*, 1984; Bowman *et al*, 1990; Wrigley *et al*, 1991]. Así mismo se han aplicado a diferentes lenguajes de programación desde lenguajes ensambladores hasta lenguajes de cuarta generación [Halstead, 1977; Martínez, 1994], en donde se ha demostrado, empíricamente, que siguen siendo aplicables para estos lenguajes.

Este estudio trata de determinar si las Métricas de Halstead son aplicables a los lenguajes manipuladores de bases de datos (LMBD).

1.2. Objetivo del Estudio

El objetivo principal de este estudio es responder a las siguientes preguntas de investigación:

1. ¿Es la métrica de longitud del programa propuesta por Halstead un buen estimador de la longitud de los programas escritos en los LMBD?

2. ¿Afecta el tamaño y la función de un programa en la estimación de la longitud de programas escritos en los LMBD?
3. ¿Es el nivel del lenguaje de los LMBD mayor que el nivel de los lenguajes de tercera generación y menor que el nivel del Inglés Prosaico?

1.3. Justificación del Estudio

En la Ciencia del Software se propone un estimador \hat{N} para la longitud de un programa, el cual produce buenas estimaciones del tamaño de los programas escritos en lenguajes de tercera [Halstead, 1977] y cuarta generación [Martínez, 1994]. Por esta razón, este estudio trata de determinar si el estimador de la longitud de un programa es buen estimador para los programas escritos en un LMBD.

También se ha demostrado que se realizan diferentes estimaciones según el tamaño o función de un programa [Verner *et al.*, 1992]. Tomando en cuenta estos resultados, en este estudio se clasifican los programas de acuerdo al tamaño y a la función que realizan para determinar si las estimaciones en los programas de los LMBD son diferentes según su clasificación.

Halstead también propuso una clasificación para varios lenguajes de programación, en la cual se incluían lenguajes ensambladores y de tercera generación, además del Inglés Prosaico. En esta clasificación se ordenaron los lenguajes según su nivel del lenguaje, donde el nivel de los lenguajes de tercera generación es, lógicamente, mayor que el nivel de los lenguajes ensambladores y menor que el Inglés Prosaico.

Se ha demostrado que el nivel del lenguaje, para lenguajes de cuarta generación, es mayor que los lenguajes de tercera generación [Martínez, 1994]. En esta tesis se obtiene

el nivel del lenguaje de los LMBD para determinar en qué lugar de la clasificación anterior se ubican, el cual se espera que sea mayor a los lenguajes de tercera generación y menor que el Inglés Prosaico, y similar al nivel de los lenguajes de cuarta generación.

1.4. Limitaciones del Estudio

Aunque en la Ciencia del Software [Halstead, 1977] se definen métricas para diferentes características de un programa como: longitud, vocabulario, volumen, nivel de programa, tiempo y esfuerzo de programación, nivel del lenguaje, entre otras, en este estudio sólo se aplican las métricas de longitud de programa y nivel del lenguaje a los programas escritos en los LMBD seleccionados.

Otra limitación de este estudio es que sólo se aplicarán las Métricas de Halstead a dos LMBD (PROGRESS y ORACLE), por lo tanto, los resultados obtenidos en esta investigación solo son válidos para los LMBD y métricas seleccionadas para el estudio.

CAPITULO 2

ANTECEDENTES

2.1. Introducción

El objetivo principal de este estudio es aplicar las Métricas de Halstead a lenguajes manipuladores de bases de datos. Por esta razón es necesario dar una explicación del lugar donde se ubican las Métricas de Halstead dentro del campo de las métricas de software.

Este capítulo se conducirá de la siguiente manera. En la sección 2.2 se dará una explicación del concepto de software, su definición, clasificación y además se explicarán las generaciones de los lenguajes de programación. En la sección 2.3 se definirá la Ingeniería del Software, los tres paradigmas que la componen y la unificación de los mismos; posteriormente, se explicarán los problemas relacionados al desarrollo del software y las causas de dichos problemas; también se explicarán los problemas asociados con las metodologías del desarrollo de software. En la sección 2.4 se explicarán las métricas de software, su clasificación, y un enfoque para su implementación. En la sección 2.5 se explicarán las métricas de calidad del software y su clasificación. En la sección 2.6 se dará un resumen del capítulo.

2.2. Software

2.2.1. Concepto de Software

El concepto de *software* es muy utilizado en nuestros días por cualquier persona que esté relacionada con el uso de las computadoras. Existen muchas conceptualizaciones sobre el software; sin embargo no existe una definición formal que explique completamente el concepto de software. No obstante, todas dan una idea general sobre su significado, por ejemplo el software es: i) “programas o conjunto de instrucciones que dirigen las operaciones de procesamiento de información realizadas por el *hardware*” [Athey *et al.*,1988, p. 15]; ii) “programas y procedimientos de computadora correspondientes a la operación de un sistema de información” [O’Brien, 1990, p. 655].

El software de computadora es información que existe en dos formas básicas: componentes no ejecutables en la máquina y componentes ejecutables en la máquina [Pressman, 1993, p. 14]. Los componentes de software se crean mediante una serie de traducciones que hacen corresponder los requisitos del cliente con un código ejecutable en la máquina.

El software es un elemento que es lógico, en lugar de físico, por lo tanto, el software tiene características considerablemente distintas a las de cualquier otro producto:

1. El software se desarrolla, no se fabrica en un sentido clásico.
2. El software no se estropea con el tiempo (en teoría, la curva de fallos para el software tendría la forma que se presenta en la figura 1).

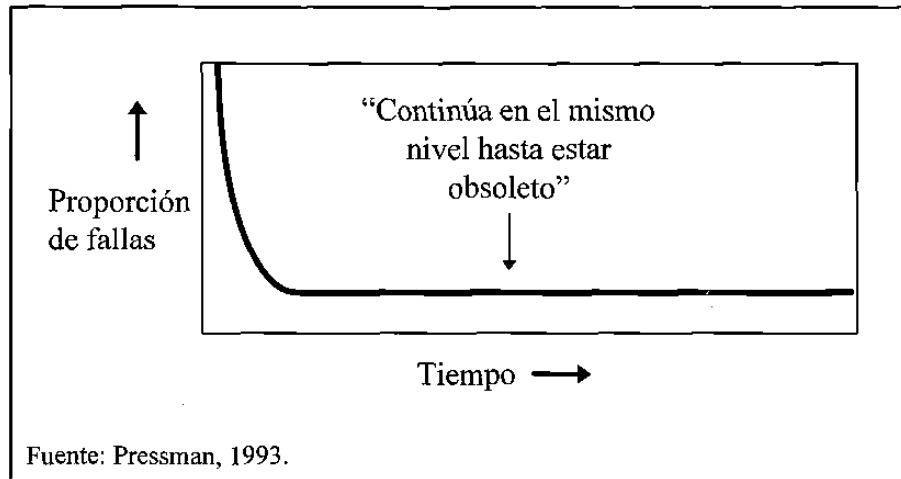


Figura 1. Curva de Fallas del Software (Idealizada)

3. La mayoría del software se construye a la medida en vez de ensamblar componentes diferentes

En la actualidad la característica 3 está desapareciendo debido a la reutilización del software. Con la reutilización del software un programador modifica la funcionalidad de un programa reemplazando los elementos u objetos antiguos por los nuevos objetos o simplemente incorporando nuevos objetos a la aplicación [Winblad *et al*, 1993, p.49].

2.2.2. Clasificación del Software

El software se puede clasificar de la siguiente forma [O'brien, 1990, p. 174]:

- *Software del Sistema*. Programas que administran y soportan los recursos y operaciones de un sistema de computadora, así como sus actividades de procesamiento de información.

- *Software de Aplicación.* Programas que dirigen el desempeño de un uso particular, o aplicación, de computadoras para satisfacer las necesidades de procesamiento de información de los usuarios finales.

2.2.3. Lenguajes de Programación

Los *lenguajes de programación* permiten a los programadores y a los usuarios finales desarrollar los programas de instrucciones que son ejecutados por una computadora. Los lenguajes de programación son una de las categorías principales del software del sistema que requieren del uso de programas traductores de lenguaje para convertir instrucciones del lenguaje de programación en instrucciones del lenguaje de máquina. Cada lenguaje tiene su propio vocabulario, gramática y usos únicos.

Los lenguajes de programación, al igual que el hardware de las computadoras han evolucionado a través de distintas generaciones [Johnson, 1990, p.220]. La figura 2 muestra las cuatro generaciones de lenguajes de programación.

Lenguaje de programación	Característica
Lenguajes de cuarta generación	Uso de declaraciones naturales y no procedimentales
Lenguajes de alto nivel	Uso de declaraciones parecidas al lenguaje inglés y notación aritmética
Lenguajes ensambladores	Uso de instrucciones codificadas simbólicamente
Lenguajes de máquina	Uso de instrucciones en código binario
Fuente: O'brien, 1990	

Figura 2. Las Cuatro Generaciones de los Lenguajes de Programación

Los *lenguajes de máquina* son el nivel más elemental de los lenguajes de programación y corresponden a la primera generación de lenguajes de programación. En las primeras etapas del desarrollo computacional, todas las instrucciones de los programas tenían que ser escritas utilizando códigos binarios únicos para cada computadora. La programación en lenguaje de máquina requiere de especificar los lugares de almacenamiento para cada instrucción y elemento de datos utilizado. Al igual que en muchas instrucciones de computadora, estas instrucciones consisten en un código de operación, el cual especifica lo que será hecho, y un operando, el cual especifica la dirección de los datos o dispositivos a ser operados. Estos requerimientos hacen que la programación en lenguajes de máquina sea una tarea difícil y propensa a errores.

Los *lenguajes ensambladores* correspondientes a la segunda generación de lenguajes de programación fueron desarrollados para reducir las dificultades al escribir los programas en lenguaje de máquina. El uso de lenguajes ensambladores requiere de programas traductores de lenguaje llamados ensambladores, los cuales permiten a la computadora convertir las instrucciones de tales lenguajes a instrucciones de máquina.

En los lenguajes ensambladores, las abreviaciones alfabéticas, que son más fáciles de recordar, son utilizadas en lugar de los números de direcciones actuales de los datos. Esto simplifica la programación, ya que los programadores no necesitan conocer las localizaciones de memoria de los datos e instrucciones. Sin embargo, los lenguajes ensambladores son orientados a la máquina, ya que cada instrucción del lenguaje ensamblador corresponde a una instrucción del lenguaje de máquina.

Los lenguajes de programación de tercera generación (3GL's) son los *lenguajes de alto nivel* también conocidos como lenguajes compiladores. Las instrucciones de los lenguajes de alto nivel son llamadas declaraciones y se asemejan al lenguaje humano o a la notación estándar de las matemáticas.

En los 3GL's, la sintaxis (reglas gramaticales, de puntuación y vocabulario) y la semántica (significados) de cada declaración no reflejan el código interno de cualquier computadora. Las declaraciones individuales de los lenguajes de alto nivel son macro instrucciones; esto es, cada declaración individual genera varias instrucciones de máquina cuando es traducido a lenguaje de máquina por los programas traductores de los lenguajes de alto nivel llamados compiladores o intérpretes.

Los lenguajes de alto nivel son más fáciles de aprender y entender que los lenguajes ensambladores debido a que tienen reglas, formas y sintaxis menos rígidas, lo cual reduce el potencial de error. La principal desventaja de estos lenguajes es que son menos eficientes que los programas de lenguajes ensambladores y requieren más cantidad de tiempo de computadora para realizar la traducción a instrucciones de máquina. Los lenguajes de alto nivel que han sido más utilizados son COBOL para programas de aplicaciones de negocios; PASCAL, BASIC para microcomputadoras de usuarios finales; FORTRAN para aplicaciones científicas y de ingeniería; Lenguaje C para desarrollar software de sistemas y de aplicación.

El término *lenguajes de cuarta generación* se usa para describir una variedad de lenguajes de programación que son menos procedimentales que los lenguajes anteriores. Estos lenguajes son llamados lenguajes de cuarta generación (4GL's) para diferenciarlos de los lenguajes de máquina (primera generación), lenguajes ensambladores (segunda generación) y lenguajes de alto nivel (tercera generación o 3GL's).

La mayoría de los lenguajes de cuarta generación permiten a los usuarios especificar los resultados que ellos quieren, mientras que la computadora determina la secuencia de instrucciones que se realizará para lograr esos resultados. Los programadores y usuarios no tienen que emplear mucho tiempo en desarrollar secuencias de instrucciones que la computadora tiene que seguir para realizar las tareas de procesamiento de información. De esta forma, los 4GL's han ayudado a simplificar el proceso de programación.

La facilidad de uso de los 4GL's se ha ganado a costa de la pérdida en la flexibilidad. El código en lenguaje de máquina producido por un 4GL es frecuentemente mucho menos eficiente (en términos de velocidad de procesamiento y cantidad de almacenamiento requerido) que el código producido por un lenguaje de alto nivel. Las aplicaciones programadas en un 4GL por lo general no proporcionan tiempos de respuesta razonables principalmente cuando estas aplicaciones son muy grandes. Sin embargo los 4GL han mostrado un gran éxito en las aplicaciones de los usuarios finales que no tienen un gran volumen de información para procesar.

Los lenguajes de cuarta generación pueden ser clasificados de la siguiente forma [Johnson, 1990, p. 227]:

1. *Lenguajes de consulta.* Estos lenguajes están diseñados para extraer información de un sistema manejador de bases de datos (SMBD) por los usuarios finales sin necesidad de preparar un programa completo. La principal característica es que pueden hacer que los usuarios finales extraigan datos de la computadora sin asistencia profesional. Algunos de los lenguajes de consulta comunes son SQL, QBE y DATAQUERY.
2. *Software para el desarrollo integrado.* Estos lenguajes constan de un SMBD, un diccionario de datos integrado, lenguajes de consulta, generadores de reporte y generadores de aplicaciones. Este tipo de lenguajes es utilizado en ambiente *mainframe*.
3. *Software para centros de información.* Estos lenguajes son productos que tienen su propio sistema manejador de base de datos, tienen algunas capacidades de apoyo para las decisiones y tienen un diccionario de datos activo. Algunos ejemplos de este tipo de lenguajes son FOCUS y NOMAD 2.

4. *SMBD para mini-computadoras y computadoras personales.* Estos son similares al software para el desarrollo integrado de productos sólo que son desarrollados para este tipo de computadoras. Algunos ejemplos de este tipo de lenguajes son ORACLE, POWERHOUSE, PROGRESS, DBASE y FOXPRO.

2.2.4. Lenguajes Manipuladores de Bases de Datos.

Un *lenguaje manipulador de bases de datos* (LMBD) puede entenderse como un programa que tiende un puente entre las estructuras de un archivo que guardan los datos y las estructuras de datos que representan las necesidades de información de los usuarios [Senn, 1992, p. 673].

Existen tres criterios para evaluar el funcionamiento de los LMBD [Kruglinski,1985]:

1. *Tiempo de acceso:* Es el tiempo que transcurre entre una petición de datos y su aparición en pantalla.
2. *Velocidad de procesamiento:* incluye la velocidad y tipo de acceso al disco, así como la indexación y clasificación de la base de datos.
3. *Capacidad de almacenamiento de datos:* El tamaño de la base de datos está limitado por el sistema operativo y la capacidad del disco de la computadora.

Las características necesarias que debe tener un lenguaje manipulador de base de datos son:

- Diccionario de datos
- Medios de consulta
- Generador de informes
- Compatibilidad con archivos de otros programas
- Capacidad de reestructuración
- Manipulación efectiva de errores
- Buena documentación y apoyo al software

Además de tener las características necesarias anteriores, se desea que un LMBD cumpla con las siguientes características:

- Archivos múltiples
- Edición de pantalla completa
- Generación de formatos de presentación
- Seguridad
- Capacidad multiusuario

2.3. Ingeniería del Software

2.3.1. Elementos Clave de la Ingeniería del Software

La *Ingeniería del Software* [Pressman, 1993, p. 25] es una disciplina para el desarrollo del software que combina métodos completos para todas las fases de desarrollo del software, mejores herramientas para automatizar estos métodos, bloques de construcción más potentes para la implementación del software, mejores técnicas para la garantía de la calidad del software y una filosofía predominante para la coordinación, control y administración.

La Ingeniería del Software surge de la ingeniería de sistemas y de hardware. Abarca un conjunto de tres elementos clave (métodos, herramientas y procedimientos) que facilitan el control del desarrollo del software y suministran a los que practiquen dicha ingeniería las bases para construir software de alta calidad en forma productiva.

Los *métodos* de la Ingeniería del Software indican “cómo” construir técnicamente el software. Abarcan un amplio espectro de tareas que incluyen la planeación y estimación de proyectos, análisis de los requisitos del sistema y del software, diseño de estructuras de datos, arquitectura de programas y procedimientos algorítmicos, codificación, prueba y mantenimiento. Estos métodos introducen frecuentemente una notación especial orientada a un lenguaje además de un conjunto de criterios para la calidad del software.

Las *herramientas* de la Ingeniería del Software suministran un soporte automático o semiautomático para 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 Asistida por Computadora* (CASE).

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 a coordinar los cambios y las directrices que ayudan a los administradores del software a evaluar el progreso.

2.3.2. Paradigmas de la Ingeniería del Software

La Ingeniería del Software está compuesta por una serie de pasos que abarcan los métodos, las herramientas y procedimientos antes mencionados. Estos pasos se les denomina frecuentemente *paradigmas de la Ingeniería del Software*.

La elección de un paradigma para la Ingeniería del Software se lleva a cabo de acuerdo con la naturaleza del proyecto y de la aplicación, los métodos y herramientas a utilizar y los controles y entregas requeridos. Tres son los paradigmas que se han tratado ampliamente: El ciclo de vida clásico, construcción de prototipos y el modelo en espiral.

2.3.2.1 El Ciclo de Vida Clásico

La figura 3 ilustra el paradigma del *ciclo de vida clásico* para la Ingeniería del Software. Este paradigma exige un enfoque sistemático y secuencial del desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento. Modelado a partir del ciclo convencional de una ingeniería, el ciclo de vida clásico abarca las siguientes actividades:

1. *Ingeniería y análisis del sistema.* Debido a que el software es parte de un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y luego asignando algún subconjunto de estos requisitos al software. Este planteamiento del sistema es esencial cuando el software debe interrelacionarse con otros elementos, tales como hardware, personas y bases de datos.
2. *Análisis de los requisitos del software.* El proceso de recopilación de los requisitos se centra e intensifica especialmente para el software. Para comprender la naturaleza de los programas que hay que construir, el ingeniero del software, o analista de sistemas, debe comprender el ámbito de la información del software, así como la

función, el rendimiento y las interfaces requeridas. Los requisitos, tanto del sistema como del software, se documentan y revisan con el cliente.

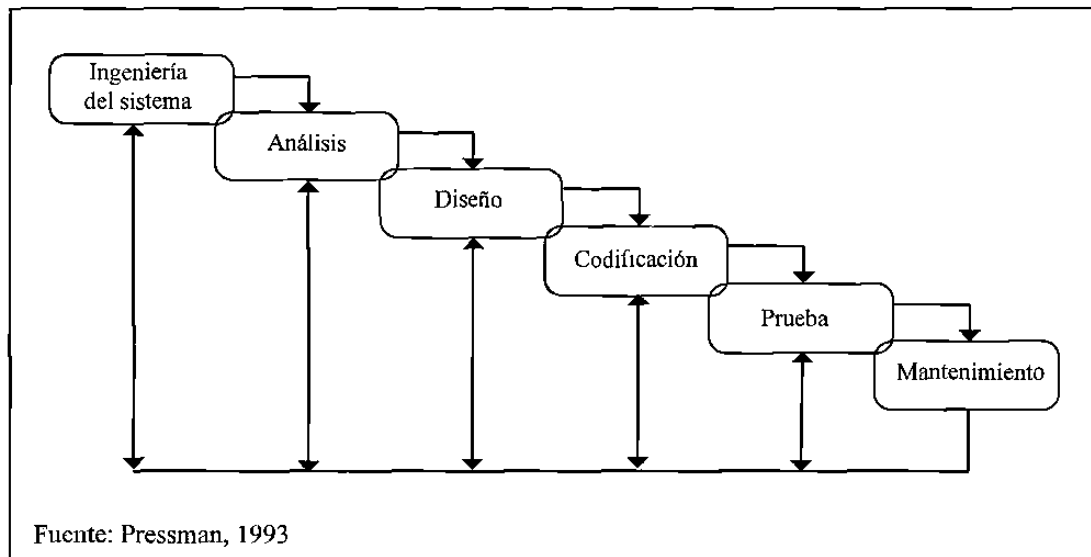


Figura 3. El Ciclo de Vida Clásico

3. *Diseño*. El diseño del software es realmente un proceso multipaso que se enfoca sobre cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software que pueda ser establecida de forma que obtenga la calidad requerida antes de que comience la codificación. Al igual que los requisitos, el diseño se documenta y forma parte de la configuración del software.
4. *Codificación*. El diseño debe traducirse en una forma legible para la máquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada, la codificación puede realizarse mecánicamente.

5. *Prueba*. Una vez que se ha generado el código, comienza la prueba del programa. La prueba se centra en la lógica interna del software, asegurando que todas las sentencias se han probado, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.
6. *Mantenimiento*. El software, indudablemente, sufrirá cambios después de que se entregue al cliente. Los cambios ocurrirán debido a que se hayan encontrado errores, a que el software deba adaptarse a cambios del entorno externo, o debido a que el cliente requiera ampliaciones funcionales o del rendimiento. El mantenimiento del software aplica cada uno de los pasos precedentes del ciclo de vida a un programa existente.

2.3.2.2 Construcción de Prototipos.

Otro de los paradigmas de la Ingeniería del Software es la *construcción de prototipos*. Este es un proceso que facilita la creación de un modelo del software a construir. El modelo tomará una de las tres formas siguientes:

1. Un prototipo en papel o un modelo basado en computadora que describa la interacción hombre-máquina, de forma que facilite al usuario la comprensión de cómo se producirá tal interacción.
2. Un prototipo que implemente algunos subconjuntos de la función requerida del programa deseado.
3. Un programa existente que ejecute parte o toda la función deseada, pero que tenga otras características que deban ser mejoradas en el trabajo de desarrollo.

La figura 4 muestra la secuencia de pasos del paradigma de construcción de prototipos.

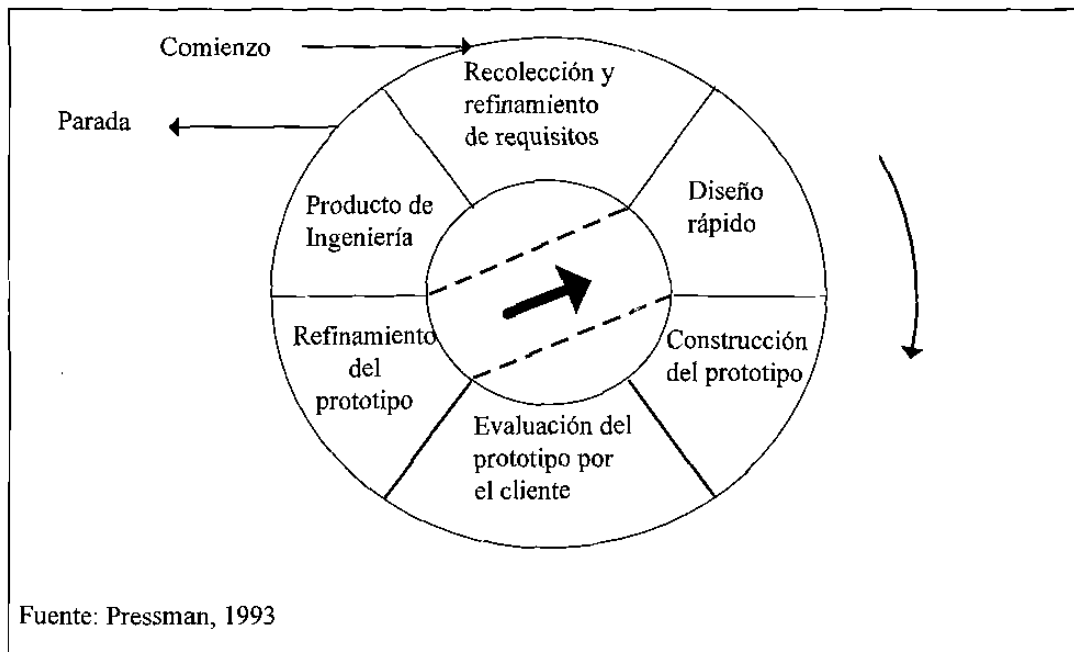


Figura 4. Creación de Prototipos

La construcción de prototipos comienza con la recolección de los requisitos. El técnico y el cliente se reúnen para definir los objetivos globales para el software, identifican todos los requisitos definidos y perfilan las áreas donde será necesario una mayor definición. Luego se propone un “diseño rápido”. El diseño rápido se enfoca sobre la representación de los aspectos visibles del software para el usuario.

Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente y el usuario, y se utiliza para refinar los requisitos del software a desarrollar. Se produce un proceso interactivo en el que prototipo es “refinado” 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 mecanismo para identificar los requisitos del software. Si se va a construir un prototipo que funcione, el realizador intenta hacer uso de fragmentos de programas existentes o aplica herramientas que faciliten la rápida generación de programas que funcionen.

2.3.2.3 Modelo en Espiral

El paradigma del modelo en espiral, representado mediante la espiral en la figura 5, define cuatro actividades principales:

1. *Planeación*: determinación de objetivos, alternativas y restricciones.
2. *Análisis de riesgo*: análisis de alternativas e identificación / resolución de riesgos.
3. *Ingeniería*: desarrollo del producto de “siguiente nivel”.
4. *Evaluación del cliente*: valoración de los resultados de la ingeniería.

Un aspecto intrigante de este modelo se hace evidente cuando consideramos su dimensión radial. Con cada iteración alrededor de la espiral se construyen sucesivas versiones del software, cada vez más completas. Durante la primera vuelta se definen los objetivos, las alternativas y restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgos indica que hay una incertidumbre en los requisitos, se pueden usar la creación de prototipos en la actividad de ingeniería para dar asistencia tanto al encargado del desarrollo como al cliente.

El cliente evalúa el trabajo de ingeniería y sugiere modificaciones. En base a los comentarios del cliente se produce la siguiente fase de planeación y análisis de riesgo.

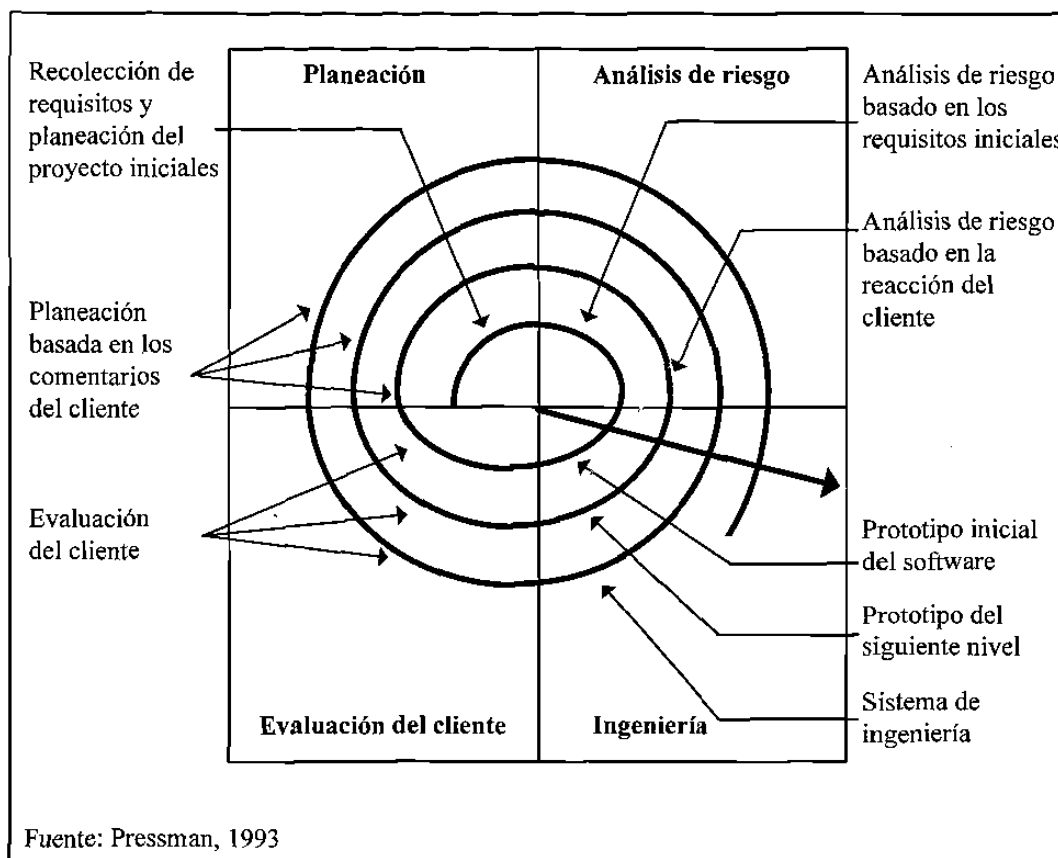


Figura 5. El Modelo en Espiral

En cada ciclo alrededor de la espiral, la culminación del análisis de riesgo resulta en una decisión de seguir o no seguir. Si los riesgos son demasiado grandes, se puede dar por terminado el proyecto, si no es así, se sigue avanzando alrededor del camino en espiral, el cual lleva a los desarrolladores a un modelo más completo del sistema, y al final, al propio sistema operacional. Cada vuelta alrededor del espiral requiere ingeniería, que se puede llevar a cabo mediante el enfoque de ciclo de vida clásico o de la creación de prototipos.

2.3.3. Unificación de los Paradigmas de la Ingeniería del Software.

Cada uno de los paradigmas de la Ingeniería del Software tiene características que lo hacen adecuado para el desarrollo de aplicaciones dependiendo de las propiedades mismas de la aplicación.

El ciclo de vida clásico se utiliza para el desarrollo de aplicaciones donde los requerimientos del software están bien definidos, cuando el número de entradas y salidas son muchas, es decir el tamaño del proyecto es grande; también se utiliza cuando se tiene tiempo suficiente para el desarrollo y cuando el personal de desarrollo tiene experiencia en aplicaciones similares.

La construcción de prototipos se utiliza cuando existen condiciones únicas de la aplicación donde los encargados del desarrollo tienen poca experiencia, cuando los requerimientos del sistema no están bien definidos, cuando los riesgos y costos de cometer un error son altos y cuando el tamaño del proyecto es pequeño.

El modelo en espiral se utiliza con el fin de aprovechar las mejores características tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo el análisis del riesgo; utiliza la creación de prototipos para la disminución de riesgos en cualquier etapa de la evolución del producto, y además mantiene el enfoque sistemático correspondiente a los pasos establecidos por el ciclo de vida clásico incorporándolos a un marco de trabajo interactivo.

Independientemente del paradigma de ingeniería del software elegido, el proceso de desarrollo de software contiene tres fases genéricas: definición, desarrollo y mantenimiento. Estas tres fases se encuentran en todos los desarrollos de software, sin importar el área de aplicación, el tamaño del proyecto o su complejidad.

2.3.3.1. Fase de Definición

La *fase de definición* se centra sobre el “qué”. Esto es, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué interfaces han de establecerse, qué restricciones de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto. Por lo tanto, se identifican los requisitos clave del sistema y del software. Los métodos varían dependiendo del paradigma, pero de alguna forma se producirán tres pasos:

1. *Análisis del sistema*. Consiste en definir el papel de cada elemento de un sistema informático, asignando finalmente al software el papel que va a desempeñar.
2. *Planeación del proyecto del software*. Una vez establecido el ámbito del software, se analizan los riesgos, se asignan los recursos, se estiman los costos, se definen tareas y se planifica el trabajo.
3. *Análisis de requisitos*. El ámbito establecido para el software proporciona la dirección a seguir, pero antes de comenzar a trabajar es necesario disponer de una información más detallada del ámbito de información y de función del software.

2.3.3.2. Fase de Desarrollo

La *fase de desarrollo* se centra en el “cómo”. Esto es, durante esta fase, el que desarrolla el software intenta descubrir cómo han de diseñarse las estructuras de datos y la arquitectura del software, cómo han de implementarse los detalles procedimentales, cómo ha de traducirse el diseño a un lenguaje de programación y cómo ha de realizarse la prueba. Los métodos aplicados durante la fase de desarrollo varían, pero de alguna forma se producirán tres pasos concretos:

1. *Diseño del software.* El diseño traduce los requisitos del software a un conjunto de representaciones (algunas gráficas y otras tabulares o basadas en un lenguaje) que describen la estructura de los datos, la arquitectura, el procedimiento algorítmico y las características de la interfaz.
2. *Codificación.* Las representaciones del diseño deben ser traducidas a un lenguaje artificial, dando como resultado unas instrucciones ejecutables por la computadora. El paso de la codificación es el que lleva a cabo esa traducción.
3. *Prueba del software.* Una vez que el software ha sido implementado en una forma ejecutable por la máquina, debe ser probado para descubrir los defectos que puedan existir en la función, en la lógica y en la implementación.

2.3.3.3. Fase de Mantenimiento

La *fase de mantenimiento* se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas por la evolución del entorno de software y a las modificaciones debidas a los cambios de los requisitos del cliente dirigidos a reforzar o ampliar el sistema.

La fase de mantenimiento vuelve a aplicar los pasos de las fases de definición y de desarrollo, pero en el contexto del software ya existente. Durante la fase de mantenimiento se encuentran tres tipos de cambios:

1. *Corrección.* El mantenimiento correctivo consiste en cambiar el software para corregir los defectos.
2. *Adaptación.* El mantenimiento adaptativo consiste en modificar el software para acomodarlo a los cambios de su entorno externo.

3. *Mejora*. El mantenimiento perfectivo amplía el software más allá de sus requisitos funcionales originales.

2.3.4. Problemas con el Desarrollo de Software

Los problemas que afectan al desarrollo del software [Pressman, 1993, p. 19] se pueden caracterizar bajo muchas perspectivas diferentes, pero los responsables de los desarrollos de software se centran sobre ciertos aspectos como: (a) la planeación y estimación de costos son frecuentemente muy imprecisas; (b) la productividad de la comunidad del software no corresponde con la demanda de sus servicios; y (c) la calidad del software no llega a ser aceptable.

Los problemas anteriormente mencionados se han producido por el propio carácter del software y por los errores de las personas encargadas del desarrollo mismo. Las principales causas de los problemas del desarrollo del software son:

1. La naturaleza lógica del software.
2. Los responsables del desarrollo del software han sido ejecutivos de medio y alto nivel, sin conocimientos del software.
3. La comunicación suele romperse debido a que se comprenden mal las características especiales del software y los problemas particulares asociados con su desarrollo.
4. Los programadores de software han tenido muy poco entrenamiento formal en las nuevas técnicas de desarrollo del software.

2.3.5. Problemas Relacionados a las Metodologías Actuales

Los ambientes de computación actuales se caracterizan por problemas de inflexibilidad, baja productividad del personal de programación y acumulación de las aplicaciones [Johnson, 1990, p. 196]. La inflexibilidad a su vez crea problemas de mantenimiento ya que cambios menores en una aplicación produce un conjunto de cambios en otros programas.

2.3.5.1. Inflexibilidad

La *inflexibilidad* en los ambientes de computación actual es el resultado de varios factores incluyendo la dependencia de los datos y la complejidad. No sólo la tecnología es compleja sino también las aplicaciones que están siendo desarrolladas. La combinación de la complejidad de la tecnología y de los problemas lleva a la adopción de metodologías de desarrollo de software que son inflexibles.

Una fuente de inflexibilidad es la dependencia de los datos. La dependencia de los datos es el resultado de formatos de archivos que están siendo utilizados en las aplicaciones de software. Muchos programas de aplicación accesan el mismo archivo. En cada programa, el formato del archivo es una parte del programa. Los cambios en un archivo requieren cambios en los formatos de cada programa que accesa al archivo. Un resultado inevitable es que los cambios menores en un programa causan una serie de cambios en otros programas.

2.3.5.2. Mantenimiento

Como resultado de la creciente complejidad del mantenimiento del software, la proporción de recursos de programación dedicados al mantenimiento crecen. En algunas

organizaciones se estima que el mantenimiento requiere el 80% del presupuesto de programación [Johnson, 1990, p. 197]. Este resultado se ilustra en la figura 6.

El énfasis en el mantenimiento de las aplicaciones de procesamiento de información tiene dos efectos:

1. Acumulación en el desarrollo de aplicaciones
2. El descuido de las aplicaciones para la toma de decisiones

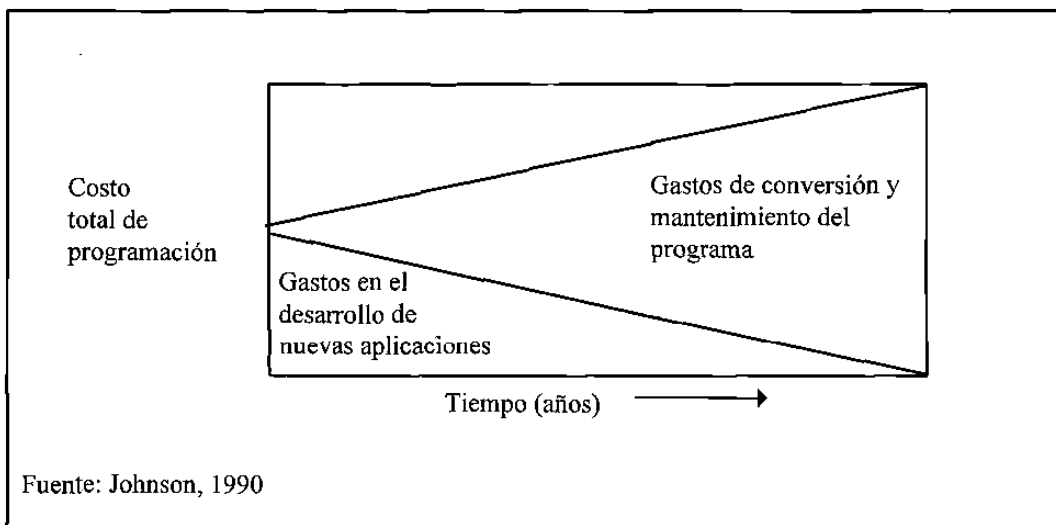


Figura 6. Mantenimiento del Software

2.3.5.3. Acumulación en el Desarrollo de Aplicaciones

La *acumulación en el desarrollo* de aplicaciones se está incrementando en la mayoría de las organizaciones debido a que la demanda de nuevas aplicaciones está creciendo más rápido que los recursos de personal de sistemas de información.

En muchos casos los usuarios finales, conscientes de esta acumulación, no hacen peticiones para nuevas aplicaciones. La acumulación de aplicaciones son especialmente frustrantes para los administradores quienes requieren aplicaciones para proveer información para la planeación y toma de decisiones. Muchas organizaciones han utilizado computadoras personales para resolver sus necesidades de información. Esto es sólo una solución parcial debido a la dificultad que tienen los usuarios finales para acceder los datos sobre las transacciones en un ambiente de procesamiento de archivos.

2.4. Métricas de Software

2.4.1. Medición y Métricas de Software

La *medición* es fundamental para cualquier disciplina de la ingeniería y en la Ingeniería del Software no es una excepción. Desgraciadamente, la medición se aleja de lo común en el mundo de la ingeniería del software. Encontramos dificultades sobre “qué medir” y “cómo evaluar” las medidas.

Hay varias razones por las cuales medir el software:

- Para indicar la calidad del producto.
- Para evaluar la productividad de la gente que desarrolla el producto.
- Para evaluar los beneficios derivados del uso de nuevos métodos y herramientas de la Ingeniería de Software.
- Para establecer una línea de base para la estimación.
- Para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

2.4.2. Clasificación de Métricas de Software

Las *métricas de software* se pueden catalogar de forma parecida a las mediciones del mundo físico, las cuales pueden englobarse en dos categorías: medidas directas y medidas indirectas. Entre las *medidas directas* del proceso de Ingeniería de Software se pueden encontrar las líneas de código producidas, la velocidad de ejecución, el número de defectos observados, etc. Entre las *medidas indirectas* se encuentran la funcionalidad, la complejidad y la calidad del software.

El campo de las métricas de software se puede clasificar tal como se muestra en la figura 7. En una dimensión tenemos la siguiente clasificación:

1. *Métricas de productividad*. Se centran en el rendimiento del proceso de ingeniería de software.
2. *Métricas de calidad*. Proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente.
3. *Métricas técnicas*. Se centran en las características del software más que en el proceso mediante el cual ha sido desarrollado el software.

En otra dimensión tenemos una segunda clasificación:

1. *Métricas orientadas a la función*. Proporcionan medidas indirectas del software y del proceso por el cual se desarrolla. Este tipo de métricas se centran en la funcionalidad o utilidad del programa.
2. *Métricas orientadas al tamaño*. Proporcionan medidas directas del software y del proceso por el cual se desarrolla. Por ejemplo, líneas de código, esfuerzo y costo.

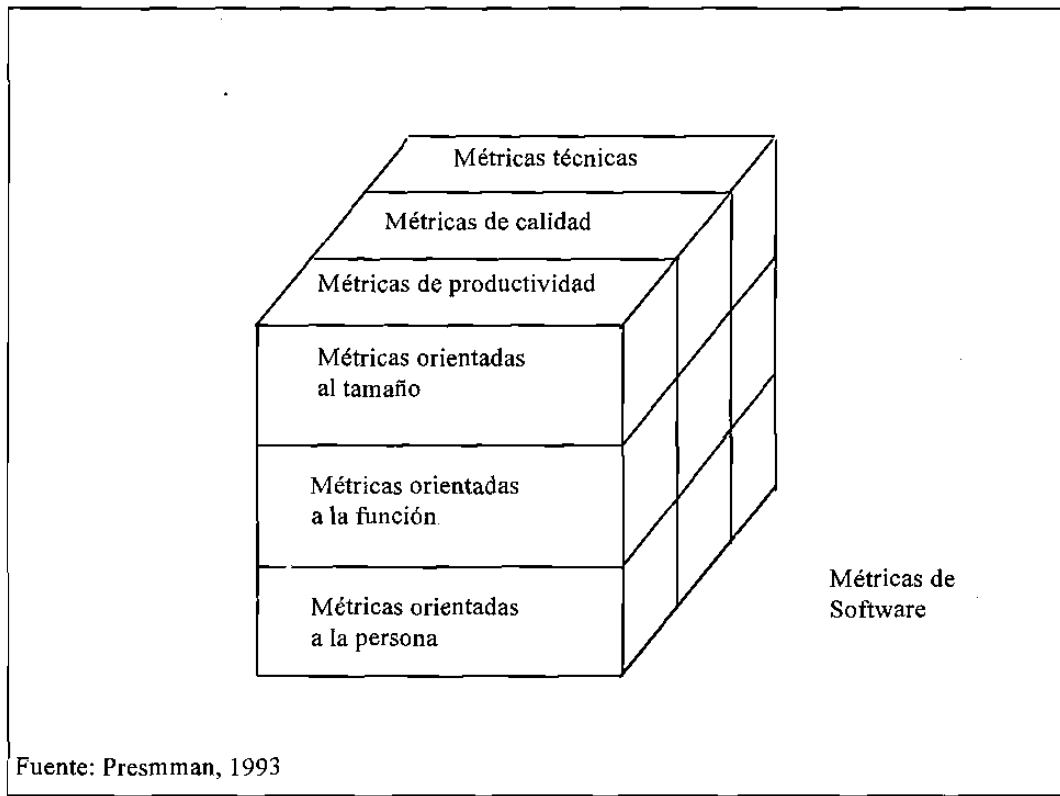


Figura 7. Clasificación de Métricas de Software

3. *Métricas orientadas a la persona.* Proporcionan información sobre la forma en que la gente desarrolla software y sobre el punto de vista humano de la efectividad de las herramientas y métodos.

2.4.3. Implementación de Métricas de Software

Se ha hecho un número de sugerencias sobre el enfoque para implementar un programa de métricas de software [Möller *et al*, 1993, p. 29]. El enfoque general es comenzar con una documentación del proceso de desarrollo de software, la cual será dinámicamente mejorada con el tiempo. Se establecen las metas con respecto a las áreas

que se quieren mejorar y se definen los objetivos deseados para la calidad sobre un período de tiempo específico. Entonces se definen las métricas, las cuales pueden ser usadas para medir el progreso periódico del proceso de desarrollo.

Los datos obtenidos de la métricas son usados como indicador de las áreas de problemas en el proceso, y se identifican las acciones para las mejoras. Además se establece un mecanismo de retroalimentación que mejora el proceso de desarrollo del producto, y da como resultado productos de alta calidad y una mejor productividad del equipo de desarrollo.

2.4.3.1. Enfoque para la Implementación de Métricas de Software

El enfoque para introducir un programa de métricas de software puede ser descrito en los siguientes siete pasos [Möller *et al*, 1993, p.47]:

1. *Proceso de desarrollo de software*. Consiste en establecer y documentar el proceso de desarrollo existente. Esto será el proceso base, el cual será medido y posteriormente mejorado.
2. *Metas*. Identificar los objetivos de la mejora, los cuales se derivan y respaldan por los objetivos estratégicos del negocio.
3. *Responsabilidad*. Identificar las responsabilidades de la administración para el programa de métricas, además de proveer el apoyo y visibilidad de cultura organizacional.
4. *Investigación inicial*. Validar los objetivos y expectativas del cliente a través de encuestas y/o evaluación del cliente.

5. *Definición de métricas.* Definir el conjunto básico de métricas para medir el progreso en el logro de las metas.
6. *Difusión.* Introducir y comunicar el programa de métricas a la organización, de tal forma que se pueda lograr una alta cooperación.
7. *Retroalimentación y proceso de mejora.* Identificar los métodos para reportar las métricas y mecanismos de retroalimentación de manera que las acciones para la mejora del proceso de desarrollo de software puedan ser determinadas e implementadas.

2.4.3.2. Problemas en la Implementación de Métricas de Software

Uno de los principales problemas en la implementación de un programa de métricas de software es la falta de aceptación. Las organizaciones de software puede que no acepten la implementación de un programa de métricas de software. Esto es resultado de una percepción de que se establecen requerimientos de control y esfuerzo para el personal de desarrollo de software sin su participación en la planeación del programa de métricas.

Algunas de las razones específicas para la falta de aceptación son:

- Las métricas pueden restringir el proceso creativo
- Las métricas crearán trabajo adicional
- Los beneficios no están claros
- El temor a ser medidos
- La dificultad para admitir que es necesaria una mejora
- La falta de apoyo de la administración
- La carencia de herramientas para obtener la métricas

2.5. Métricas de Calidad del Software

2.5.1. Calidad del Software

En los libros se han propuesto muchas definiciones de calidad de software. Una de ellas la define como: "Concordancia de los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" [Pressman, 1993, p. 576]. Esta definición hace énfasis en tres puntos importantes:

1. Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad.
2. Los estándares especificados definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería de software. Si no se siguen estos criterios, casi siempre habrá falta de calidad.
3. Existe un conjunto de requisitos implícitos que a menudo no se mencionan. Si el software se ajusta a sus requisitos explícitos pero falla en alcanzar los requisitos implícitos, la calidad del software queda en entredicho.

Los factores que afectan la calidad del software se clasifican en dos categorías:

1. Factores que pueden ser medidos directamente.
2. Factores que solo pueden ser medidos indirectamente.

La figura 8 muestra una clasificación de los factores que afectan la calidad del software propuesta por McCall [Pressman, 1993, p.577] . Estos factores se basan en tres aspectos importantes del software:

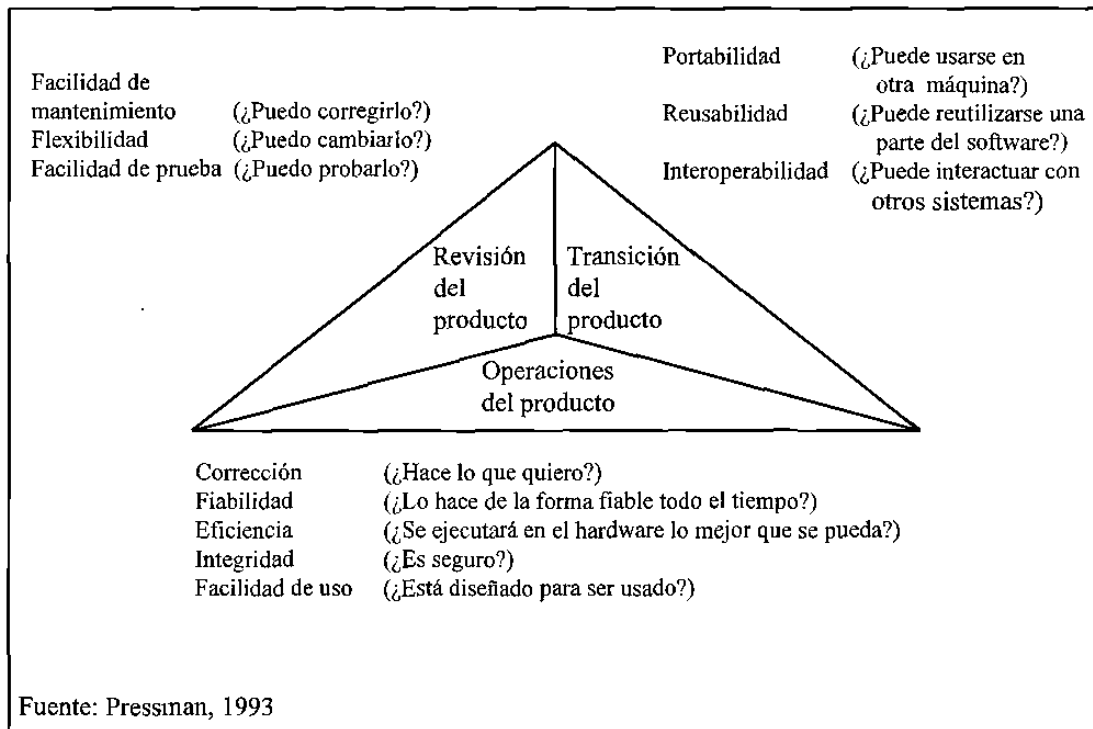


Figura 8. Factores de Calidad del Software de McCall

a) Características operativas:

- *Corrección.* El grado en que un programa satisface sus especificaciones y consigue los objetivos de la misión encomendada por el cliente.
- *Fiabilidad.* El grado en que se puede esperar que un programa lleve a cabo sus funciones esperadas con la precisión requerida.
- *Eficiencia.* La cantidad de recursos de computadora y de código requeridos por un programa para llevar a cabo sus funciones.
- *Integridad.* El grado en que puede controlarse el acceso al software o a los datos.

- *Facilidad de uso*. El esfuerzo requerido para aprender un programa, trabajar con él, preparar su entrada e interpretar su salida.

b) Capacidad de soportar los cambios:

- *Facilidad de mantenimiento*. El esfuerzo requerido para localizar y corregir un error.
- *Flexibilidad*. El esfuerzo requerido para modificar un programa operativo.
- *Facilidad de prueba*. El esfuerzo requerido para probar un programa de forma que se asegure que realiza la función requerida.

c) Adaptabilidad a nuevos entornos:

- *Portabilidad*. El esfuerzo requerido para transferir el programa desde un hardware y/o un entorno de sistemas de software a otro.
- *Reutilización*. El grado en que un programa se puede reutilizar en otras aplicaciones.
- *Facilidad de interoperación*. El esfuerzo requerido para acoplar un sistema a otro.

Es difícil desarrollar medidas directas de los anteriores factores de calidad. Por lo tanto, se define un conjunto de métricas usadas para medir en forma indirecta los factores de calidad del software.

2.5.2. Métricas Cualitativas

Las *métricas cualitativas* son aquellas que miden el software en forma indirecta o subjetiva, entre ellas se encuentran las métricas definidas por McCall [1977] (citado por Pressman, 1993):

- *Facilidad de auditoría*. La facilidad con que se puede comprobar la conformidad de los estándares.
- *Exactitud*. La precisión de los cálculos y del control.

- *Normalización de las comunicaciones.* El grado en que se usan el ancho de banda, los protocolos y las interfaces estándar.
- *Completitud.* El grado en que se ha conseguido la total implementación de las funciones requeridas.
- *Concisión.* Lo compacto que es el programa en términos de líneas de código.
- *Consistencia.* El uso de un diseño uniforme y de técnicas de documentación a lo largo de un proyecto de desarrollo de software.
- *Estandarización de los datos.* El uso de estructuras de datos y de tipos estándar a lo largo de todo el programa.
- *Tolerancia a errores.* El daño que se produce cuando el programa encuentra un error.
- *Eficiencia en la ejecución.* El rendimiento en tiempo de ejecución de un programa.
- *Facilidad de expansión.* El grado en que se puede expandir el diseño arquitectónico, de datos o procedimental.
- *Generalidad.* La amplitud de aplicación potencial de los componentes del programa.
- *Independencia del hardware.* El grado en que el software es independiente del hardware sobre el cual opera.
- *Instrumentación.* El grado en que el programa muestra su propio funcionamiento e identifica errores que aparecen.
- *Modularidad.* La independencia funcional de los componentes del programa.
- *Facilidad de operación.* La facilidad de operación de un programa.
- *Seguridad.* La disponibilidad de mecanismos que protejan los programas y datos.
- *Autodocumentación.* El grado en que el código fuente proporciona documentación significativa.
- *Independencia del sistema de software.* El grado en que el programa es independiente de características no estándar del lenguaje de programación, del sistema operativo y de otras restricciones del entorno.
- *Facilidad de traza.* La posibilidad de seguir la pista a la representación del diseño o de los componentes reales del programa hacia atrás, hacia los requisitos.
- *Formación.* El grado en que el software ayuda a permitir que nuevos usuarios apliquen el sistema.

2.5.3. Métricas Cuantitativas

Las *métricas cuantitativas* son aquellas que miden el software en forma directa u objetiva. En las métricas cuantitativas del software encontramos, entre otras, la *Ciencia del Software* [Halstead, 1977].

La Ciencia del Software propone las primeras leyes analíticas de la Informática, asigna leyes cuantitativas al desarrollo de software de computadora [Pressman, 1993, p. 600]. La teoría de Halstead se deriva de una suposición fundamental: "El cerebro humano sigue un conjunto de reglas de las que nunca se ha tenido consciencia..." [Halstead, 1977, p.15].

La Ciencia del Software utiliza un conjunto de primitivas de medida que se pueden obtener una vez que se ha generado el código, o estimar una vez que se ha terminado el diseño. Estas medidas son las siguientes:

- Número de operadores distintos que aparecen en un programa
- Número de operandos distintos que aparecen en un programa
- Número total de ocurrencias de los operadores
- Número total de ocurrencias de los operandos

Halstead utiliza las primitivas de las medidas para desarrollar expresiones para la longitud total de un programa, el mínimo volumen potencial de un algoritmo, el volumen real, el nivel del programa, el nivel del lenguaje y otras características tales como el esfuerzo de desarrollo, el tiempo de desarrollo e incluso el número previsto de fallos en el software.

2.6. Resumen

En este capítulo se explicaron los conceptos de software, Ingeniería del Software y métricas de software. Así mismo se dio una clasificación de las métricas del software dentro de las cuales se ubican las Métricas de Halstead. En el siguiente capítulo se explicarán en forma más detallada las Métricas de Halstead.

CAPITULO 3

METRICAS DE HALSTEAD

3.1. Introducción

En el capítulo anterior se ubicaron las métricas de Halstead dentro de las métricas de calidad del software. En este capítulo se explicarán en forma más detallada la Ciencia del Software [Halstead, 1977].

Este capítulo se conducirá de la siguiente forma. En la sección 3.2 se dará una breve explicación de la Ciencia del Software de Halstead, las críticas que ha recibido y sus aplicaciones. En la sección 3.3 se dará una definición de las métricas básicas de la Ciencia del Software. En la sección 3.4 se explicarán los estimadores para la longitud del programa, volumen potencial, nivel y dificultad del programa, tiempo y esfuerzo de programación. En la sección 3.5 se definirá el nivel del lenguaje para un programa y su estimador. En la sección 3.6 se dará un resumen del capítulo.

3.2. La Ciencia del Software

La *Ciencia del Software* tiene su interés en los algoritmos y su implementación, ya sea como programas de computadora o como instrumentos de comunicación humana [Halstead, 1977, p.3]. Como ciencia experimental sólo trata con aquellas propiedades de

los algoritmos que pueden ser medidas, ya sea directa o indirectamente, estática o dinámicamente; y con las relaciones entre estas propiedades que quedan invariantes después de ser traducidas de un lenguaje a otro.

3.2.1. Críticas a la Ciencia del Software

La Ciencia del Software ha recibido muchas críticas. Algunas de ellas sobre la forma en como se derivan las métricas [Shen *et al*, 1983], otras en como se aplican los métodos estadísticos para encontrar las relaciones entre las propiedades de un programa [Courtney *et al*, 1993] y otras basadas en las propiedades de la ciencia de la medición [Fenton, 1994]. En base a estas críticas se han propuesto métricas alternas a las que propuso Halstead [1977], algunas sobre la complejidad de un programa [Weuyker, 1988; Tian *et al*, 1995], otras para la longitud de un programa utilizando modelos estocásticos [Keller-McNulty *et al*, 1991] o utilizando líneas de código [Lokan, 1996] y otras basadas en las propiedades del software [Briand *et al*, 1996].

3.2.2. Aplicaciones de la Ciencia del Software

Aunque la Ciencia del Software ha recibido muchas críticas, ha sido utilizada con diversos propósitos. Las Métricas de Halstead [1977] se han utilizado para evaluar la estructura de software del sistema, con lo cual se mostró el poder de esta técnica para localizar errores de diseño e implementación [Henry *et al*, 1984]. La Ciencia del Software (y otras métricas sobre complejidad) también se ha aplicado a la educación, donde se demostró que los estudiantes que reciben entrenamientos sobre las métricas de complejidad desarrollan programas con poca complejidad, en poco tiempo y con gran calidad [Bowman *et al*, 1990]. Otro uso que se le ha dado a la Ciencia del Software ha sido para desarrollar un modelo que mide el tamaño de los sistemas de información, el

cual ayuda a los administradores a entender y establecer las relaciones entre la especificación de requerimientos de las unidades del sistema, el diseño y el código fuente [Wrigley *et al*, 1991].

3.3. Métricas Básicas

En la Ciencia del Software [Halstead, 1977] un programa está formado por una serie de partículas las cuales pueden ser consideradas como operadores u operandos. Los *operandos* son definidos como las variables o constantes que utiliza la implementación, mientras que los *operadores* son los símbolos que afectan el valor u orden de un operando. Con la identificación de operadores y operandos, es posible definir un número contable, por lo tanto medible, de entidades que deben estar presente en cualquier versión de un algoritmo. Estas propiedades son las *métricas básicas*, las cuales son definidas como:

$$\eta_1 = \text{número de operadores distintos} \quad (1)$$

$$\eta_2 = \text{número de operandos distintos} \quad (2)$$

$$N_1 = \text{número total de ocurrencias de los operadores} \quad (3)$$

$$N_2 = \text{número total de ocurrencias de los operandos} \quad (4)$$

De estas métricas básicas, se define el *vocabulario* η de un programa, el cual consiste en el número de partículas únicas utilizadas para construir un programa:

$$\eta = \eta_1 + \eta_2 \quad (5)$$

La *longitud* N del programa en términos del número de partículas utilizadas es:

$$N = N_1 + N_2 \quad (6)$$

Debe notarse que N está relacionado con la medición de líneas de código (LDC) para la longitud de un programa. Para programas en lenguaje de máquina donde cada línea consta de un operador y un operando $N = 2 \times \text{LDC}$.

Algunas métricas adicionales son definidas utilizando las métricas básicas, una de ellas es el *volumen*:

$$V = N \log_2 \eta \quad (7)$$

La unidad de medición del volumen es la unidad comúnmente utilizada para el tamaño: "*bits*". El volumen también puede ser interpretado como el número de comparaciones elementales utilizadas para escribir un programa de longitud N , asumiendo que un método binario de búsqueda es utilizado para seleccionar un elemento de un vocabulario de tamaño η . Debido a que un algoritmo puede ser implementado por programas diferentes pero equivalentes, un programa que está en su mínimo tamaño se dice que está en su *volumen potencial* V^* .

Cualquier programa con un volumen V es implementado en un *nivel de programa* L , el cual es definido como:

$$L = V^* / V \quad (8)$$

El rango de valores para L está entre 0 y 1, donde $L = 1$ representa un programa escrito en el nivel más alto posible. Lo inverso al nivel del programa es llamado *dificultad*:

$$D = 1 / L \quad (9)$$

Si el volumen de una implementación se incrementa, el nivel del programa disminuye y la dificultad aumenta.

De esta forma, en la práctica de programación, el uso redundante de operandos y el uso de instrucciones de control de nivel más alto incrementarán el volumen y la dificultad.

El esfuerzo requerido para implementar un programa de computación aumenta según aumenta el tamaño del programa. También requiere más esfuerzo de programación implementar un programa a un nivel bajo (de dificultad más alta) cuando es comparado con otro programa equivalente de nivel más alto (dificultad menor). De esta forma el *esfuerzo* en la Ciencia del Software es definido como:

$$E = V / L = D \times V \quad (10)$$

Las unidades de las medidas del esfuerzo E son las discriminaciones mentales elementales.

3.4. Estimadores de un Programa

En la Ciencia del Software de Halstead se definen estimadores para varias propiedades de un programa entre los cuales se encuentran los estimadores para la longitud, volumen potencial, nivel del programa, esfuerzo y tiempo de programación, los cuales se detallan en las siguientes secciones.

3.4.1. Estimador de la Longitud

En la Ciencia del Software la longitud de un programa bien estructurado está en función solo de los operadores y operandos únicos. La ecuación para la estimación de la longitud de un programa está dada por:

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \quad (11)$$

El estimador de longitud es adecuado en un cierto rango para programas escritos en lenguajes de tercera generación como Fortran, Pascal y PL/S [Shen *et al*, 1983] y en lenguajes de cuarta generación como DBASEIII y FOXPRO2 [Martínez, 1994]. Generalmente el estimador de la longitud comienza sobrestimando la longitud para programas pequeños y termina subestimándola para programas grandes como se muestra en la figura 9.

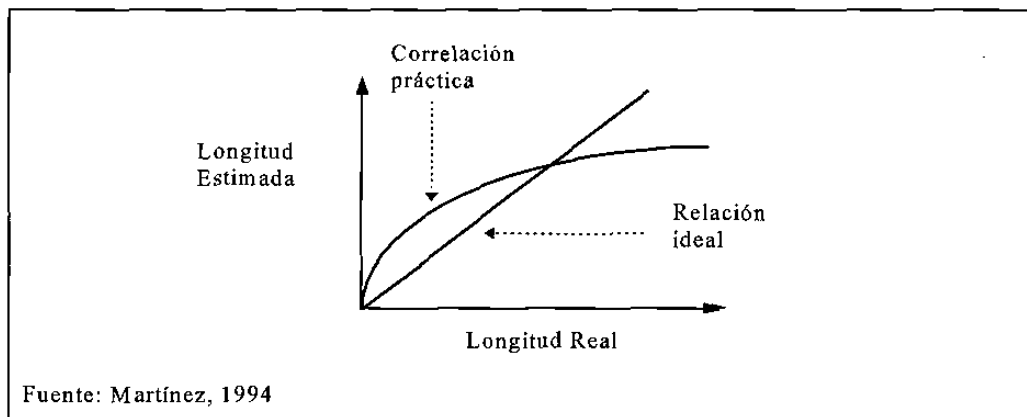


Figura 9. Relación Ideal entre Longitud Real y Longitud Estimada, y Correlación Lineal Práctica

Cuando se han utilizado las Métricas de Halstead para estimar la longitud de un programa, por lo general se han aplicado a programas completos, sin importar su modularidad. Utilizando otro tipo de métricas para estimar la longitud de un programa, se ha demostrado empíricamente que se obtienen mejores estimaciones para el tamaño de un programa si se estima el tamaño de cada uno de los módulos que componen el mismo y al final se suman las estimaciones individuales para obtener la estimación total del tamaño del programa [Verner *et al*, 1992]. Este tipo de estimación difiere de las

desarrolladas anteriormente debido a que clasifica cada módulo que compone un programa de acuerdo a la función que realiza (menú, pantallas, reportes, consultas, actualizaciones, formas, utilerías, entre otros) antes de realizar la estimación.

3.4.2. Estimador del Volumen Potencial

La forma más breve en la que un algoritmo puede ser expresado requiere de la existencia de un lenguaje en el cual la operación requerida ya esté implementada o definida, quizás como una subrutina o procedimiento. En tal lenguaje, la implementación del algoritmo requiere nada más del nombre de la función y de sus parámetros. El vocabulario de este programa consiste de dos operadores y de η_2^* operandos. Uno de los operadores es el nombre del procedimiento y el otro es un símbolo de agrupamiento que separa la lista de parámetros del nombre del procedimiento. Por lo tanto, el *volumen potencial* de un programa es:

$$V^* = (2 + \eta_2^*) \log_2 (2 + \eta_2^*) \quad (12)$$

donde η_2^* es el número de parámetros de entrada / salida para este procedimiento. Esta fórmula no se aplica a todos los programas ya que muchos de ellos no tienen una definición explícita de los parámetros de entrada / salida.

3.4.3. Estimador del Nivel de Programa

El nivel de una implementación particular depende de la razón entre el volumen potencial y el volumen actual. Debido a que el volumen potencial no está disponible, una fórmula alterna para estimar el nivel es:

$$\hat{L} = \frac{2}{\eta_1} \frac{\eta_2}{N_2} \quad (13)$$

Al observar esta fórmula vemos que la dificultad aumenta si introducimos operadores adicionales o si un operando es usado repetidamente. Cada parámetro en (13) puede ser obtenido al contar los operadores y operandos en un programa de computadora.

El volumen potencial V^* puede ser deducido utilizando (8) con $L = \hat{L}$. Esta fórmula también puede ser utilizada con (10) para determinar el estimado del esfuerzo de la Ciencia del Software.

La ecuación (8) sugiere que se pudieran tener diferentes volúmenes y niveles para un algoritmo dado, mientras que el producto de estos dos permanecen constantes. Esto es, el volumen potencial $V^* = LV$ depende solamente del algoritmo y no de las características propias de la implementación en particular. Cuando el nivel de programa \hat{L} es utilizado para estimar L , el producto LV es llamado *contenido de inteligencia*,

$$\hat{I} = LV \quad (14)$$

3.4.4. Estimador del Esfuerzo y Tiempo de Programación

Una de las principales críticas a la Ciencia del Software la constituyen las suposiciones que utiliza para derivar la métrica del tiempo de implementación [Shen *et al*, 1983]. La primer suposición es el proceso de selección que utiliza una persona para desarrollar un programa, el cuál lo aproxima a una búsqueda binaria. Aunque esta suposición concuerda con la ley de Hicks, la aplicación es bastante diferente del

ambiente en el cual Hick llevó a cabo sus experimentos. La segunda suposición es que el humano es capaz de hacer un número constante de discriminaciones mentales elementales por segundo, tal como lo sugiere el Psicólogo J. Stroud [Halstead, 1977, p. 48]. Stroud afirmó que este número S (de ahora en adelante llamado “Número de Stroud”) estaba entre 5 y 20. Esto es cuestionable, ya que dicha suposición no está totalmente aceptada por los psicólogos debido a la falta de evidencia empírica.

Como el esfuerzo E tiene por unidades de medición “número de discriminaciones mentales elementales”, el *tiempo de programación* T en segundos es:

$$T = E / S \quad (15)$$

donde S normalmente se establece como 18, ya que ésto pareció dar mejores resultados en los experimentos de Halstead cuando comparó el tiempo de predicción con el tiempo de observado (el cual incluía el tiempo para diseño, codificación y prueba).

De la ecuación (10) tenemos que el esfuerzo $E = V / L$. Si utilizamos el estimador de nivel de programa (13) y de volumen de (7), tenemos que el estimador para el esfuerzo es:

$$\hat{I} = (\eta_1 N_2 N \log_2 \eta_1) / (2\eta_2) \quad (16)$$

Por lo tanto de (15) y (16), el tiempo estimado de programación es:

$$\hat{T} = \frac{\eta_1 N_2 (\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2) \log_2 \eta_2}{2S\eta_2} \quad (17)$$

3.5. Nivel del lenguaje de Programación

Para cualquier algoritmo que es traducido de un lenguaje de implementación a otro, cuando el volumen aumenta, el nivel disminuye proporcionalmente. Como consecuencia, el producto LV es igual al volumen potencial V^* para un algoritmo dado [Halstead, 1977].

Por otra parte, cuando el lenguaje de implementación permanece constante, y el algoritmo varía, se observa una relación similar. Para este caso, a medida que el volumen potencial V^* aumenta, el nivel del programa disminuye proporcionalmente. Como consecuencia, el producto LV^* permanece constante para cualquier lenguaje. Este producto, el *nivel del lenguaje*, es denotado por λ , de tal forma que:

$$\lambda = LV^* = L^2V \quad (18)$$

Para obtener el estimador del nivel del lenguajes sustituimos L de la ecuación (18) por \hat{L} de la ecuación (13) y V de la ecuación (7), obteniendo:

$$\hat{\lambda} = [(2/\eta_1) (\eta_2 / N_2)]^2 [N \log_2 \eta] \quad (19)$$

En forma general, se espera que entre mayor sea el nivel del lenguaje mayor es el número de formas en las cuales un algoritmo pueda ser implementado. El hecho de que muchas de estas implementaciones están en diferentes niveles, produce que el valor medio de λ aumente. La varianza sobre la media también parece aumentar debido a que el nivel del lenguaje no depende en sí del lenguaje mismo sino también de la naturaleza propia del problema que está siendo programado así como de la habilidad, experiencia y estilo del programador.

Halstead aplicó las métricas de nivel del lenguaje en diversos tipos de lenguajes de programación, los cuales incluían lenguajes ensambladores y lenguajes de tercera generación, así mismo incluyó como lenguaje de referencia el Inglés Prosaico, el cual según su hipótesis debía tener el nivel del lenguaje más alto. Los resultados de estas mediciones se muestran en la tabla I. En la tabla II se pueden ver las distribuciones de frecuencia de los niveles del lenguaje para los lenguajes mencionados en la tabla I.

Tabla I

Medias y Varianzas del Nivel del lenguaje para Cinco Lenguajes

<i>Lenguaje</i>	λ	<i>Varianza</i>
Inglés	2.16	0.74
PL/I	1.53	0.92
Algol 58	1.21	0.74
Fortran	1.14	0.81
Pilot	0.92	0.43
Assembly	0.88	0.42

Fuente: Halstead, 1977

Tabla II

Distribuciones de Frecuencia de Niveles del lenguaje para Diferentes Lenguajes

λ	<i>Assembly</i>	<i>Pilot</i>	<i>Fortran</i>	<i>Algol 58</i>	<i>PL/I</i>	<i>Inglés</i>
(0.00 , 0.49)	14%	14%	14%	14%	14%	0%
(0.50 , 0.99)	57	64	43	29	29	0
(1.00 , 1.49)	14	14	29	36	21	23
(1.50 , 1.99)	14	0	7	14	14	17
(2.00 , 2.49)	0	7	0	0	21	27
(2.50 , 2.99)	0	0	0	0	0	17
(3.00 , 3.49)	0	0	0	7	0	10
(3.50 , 3.99)	0	0	7	0	7	7

Fuente: Halstead, 1977

En un estudio posterior [Martínez, 1994] se aplicaron las Métricas de Halstead para obtener el nivel del lenguaje de DBASEIII y FOXPRO2, los cuales son considerados como lenguajes de cuarta generación. Como era de esperarse estos dos lenguajes tuvieron un mayor nivel que los lenguajes de tercera generación, pero un nivel menor al Inglés Prosaico. Los resultados de este estudio se muestran en la tabla III. En la tabla IV se muestran las distribuciones de frecuencia de nivel del lenguaje para DBASEIII y FOXPRO2.

Tabla III

**Medias y Varianzas del Nivel del lenguaje para
Lenguajes de Cuarta Generación**

<i>Lenguaje</i>	λ	<i>Desviación Estándar</i>
FOXPRO2	1.97	1.70
DBASEIII	1.95	1.81

Fuente: Martínez, 1994

Tabla IV

**Distribuciones de Frecuencia de Niveles del lenguaje
para Lenguajes de Cuarta Generación**

λ	<i>DBASEIII</i>	<i>FOXPRO2</i>	λ	<i>DBASEIII</i>	<i>FOXPRO2</i>
(0 , 1]	27%	34%	(8 , 9]	4%	1%
(1 , 2]	42	26	(9 , 10]	0	0
(2 , 3]	15	21	(10 , 11]	0	0
(3 , 4]	8	10	(11 , 12]	0	0
(4 , 5]	0	5	(12 , 13]	0	0
(5 , 6]	4	1	(13 ,14]	0	0
(6 , 7]	0	0	(14 , 15]	0	1
(7 , 8]	0	1			

Fuente: Martínez, 1994

En la Ciencia del Software se establece que el nivel del lenguaje permanece constante [Halstead, 1977]. Posteriormente se observó [Shen *et al*, 1983] que en lenguajes como Cobol, Fortran y PL/S el nivel del lenguaje depende de la longitud de programa, como una función exponencialmente decreciente, como se muestra en la figura 10. Estos mismos resultados fueron encontrados al aplicar las métricas de software a los lenguajes de cuarta generación DBASEIII y FOXPRO2 [Martínez, 1994].

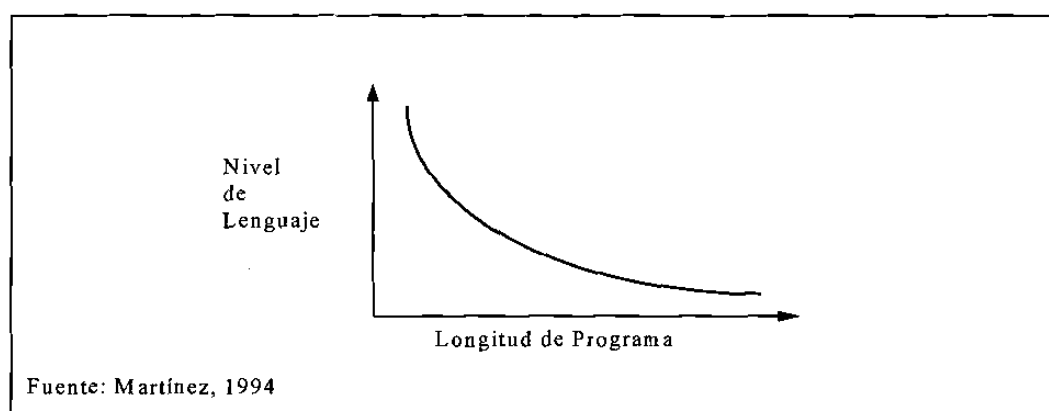


Figura 10. Relación Ideal entre λ y N .

3.6. Resumen

En este capítulo se explicó la Ciencia del Software de Halstead, la cual es el modelo teórico en la que se basa esta investigación. En el siguiente capítulo se discutirán las preguntas e hipótesis de investigación así como el enfoque que se tomará para investigar las preguntas propuestas.

CAPITULO 4

METODOLOGIA DE LA INVESTIGACION

4.1. Introducción

En el capítulo anterior se explicaron las Métricas de Halstead las cuales constituyen el marco teórico de la investigación. En este capítulo se presenta la metodología de la investigación utilizada.

Este capítulo se conducirá de la siguiente manera. En la sección 4.2 se establecerán las preguntas e hipótesis de investigación. En la sección 4.3 se explicará la selección del diseño de investigación. En la sección 4.4 se discutirá la selección de los lenguajes manipuladores de bases de datos (LMBD). En la sección 4.5 se discutirá la selección de la muestra. En la sección 4.6 se explicará el instrumento de medición. En la sección 4.7 se hará un resumen del capítulo.

4.2. Preguntas e Hipótesis de la Investigación

En el capítulo anterior se mostró que existe evidencia estadística de que el estimador de la longitud \hat{N} es un buen estimador de la longitud del programa para los lenguajes de tercera y cuarta generación. También se mostró que la exactitud de la estimación del tamaño de un programa depende del tamaño del programa [Shen *et al*,

1983; Martínez, 1994]. En base a esto podemos establecer nuestra primera pregunta de investigación: ¿Es el estimador de un programa \hat{N} un buen estimador de N en los lenguajes manipuladores de bases de datos? A partir de esta pregunta formulamos la primer hipótesis de la investigación:

H_1 : Para programas en lenguajes manipuladores de bases de datos,
 $\hat{N} = N$.

La segunda pregunta de investigación se deriva de la hipótesis de que se realizan diferentes estimaciones de la longitud de un programa dependiendo del tamaño y de la función que realiza [Verner *et al*, 1992], la cual establecimos así: ¿Afecta el tamaño y función de un programa en la estimación de la longitud para programas escritos en LMBD? De esta pregunta formulamos la segunda hipótesis de la investigación:

H_2 : Para programas en lenguajes manipuladores de bases de datos
 con diferente tamaño y función, $\hat{N} = N$.

La siguiente pregunta de investigación se origina al suponer que el nivel del lenguaje de los LMBD es mayor que los 3GL's (1.53) y menor que el nivel del Inglés Prosaico (2.16). Por lo tanto la tercer pregunta de la investigación puede establecerse como: ¿Es el nivel del lenguaje de los LMBD mayor que el nivel del lenguaje de los 3GL's y menor que el nivel del lenguaje del lenguaje natural? De esta pregunta se derivan las siguientes hipótesis:

H_3 : Para lenguajes manipuladores de bases de datos, el nivel del lenguaje
 es mayor a 1.53

H_4 : Para lenguajes manipuladores de bases de datos, el nivel del lenguaje
 es menor que 2.16

H₃: Para programas en lenguajes manipuladores de bases de datos, el nivel del lenguaje es diferente según el tamaño y la función que realizan.

4.3. Diseño de la Investigación

Debido a las características de la investigación se seleccionó el diseño *expost-facto*. La investigación no experimental o *expost-facto* es aquella que se realiza sin manipular deliberadamente variables [Hernández *et al*, 1991]. Es decir, es investigación donde no hacemos variar intencionalmente las variables independientes. Lo que hacemos en este tipo de investigación es observar los fenómenos tal y como se dan en su contexto natural, para después analizarlos.

Para la recolección de datos se utilizó como instrumento un analizador de código (ver apéndice), al cual se le da como entrada los programas en código fuente y como resultado devuelve los valores de la Métricas de Halstead.

4.4. Selección de los LMBD

Los lenguajes manipuladores de bases de datos seleccionados para este estudio fueron PROGRESS Versión 6.2 (PROGRESS6) y ORACLE Versión 7.2 (ORACLE7). Estos lenguajes fueron seleccionados por las siguientes razones:

1. Son los lenguajes manipuladores de bases de datos más vendidos y utilizados [VARB, 1995].
2. Contienen programas de ejemplo en código fuente, lo cual permite evaluarlos utilizando las Métricas de Halstead.

4.5. Selección de la Muestra

Al aplicar las Métricas de Halstead se deben eliminar las impurezas de los programas para obtener resultados más confiables [Halstead, 1977]. Por esta razón, los programas seleccionados para este estudio fueron los programas en código fuente que están incluidos como programas de ejemplo en los lenguajes PROGRESS6 y ORACLE7.

La razón para seleccionar los programas de ejemplo se debe a que estos programas son desarrollados por expertos, por lo tanto se reduce al mínimo la impurezas contenidas en ellos.

Para el lenguaje PROGRESS6 se seleccionaron un total de 957 programas de ejemplo incluidos en el paquete. El lenguaje ORACLE7 no incluía programas de ejemplo en el paquete, por lo tanto, se tuvo que seleccionar como muestra los programas que están incluidos en los manuales de usuario del paquete, que son un total de 63.

4.6. Instrumento de Medición

El instrumento de medición utilizado en este estudio para recolectar los datos fue un analizador de código, al cual se le da como entrada el programa en código fuente y produce como resultado el valor correspondiente a las Métricas de Halstead. En esta investigación se utilizó el analizador de código desarrollado para aplicar las Métricas de Halstead en los 4GL's: FOXPRO2 y DBASEIII [Martínez, 1994], al cual se le realizaron las modificaciones necesarias para utilizarlo con los LMBD PROGRESS6 y ORACLE7. En el apéndice se muestran las características operativas del analizador de código.

Para validar el analizador de código se realizaron pruebas con 30 programas pequeños (no mayores a 20 líneas), en donde se obtuvieron la Métricas de Halstead en forma manual y posteriormente se compararon con las métricas obtenidas a través del analizador de código. No hubo diferencia entre los resultados obtenidos a través del analizador y los resultados obtenidos manualmente, por lo tanto, el analizador calcula adecuadamente las Métricas de Halstead para los LMDB PROGRESS6 y ORACLE7.

4.7. Resumen

En este capítulo se presentaron las preguntas e hipótesis de la investigación, se discutió el diseño de investigación, la selección de los LMDB, la selección de la muestra así como el instrumento de medición que será empleado para determinar si las Métricas de Halstead son aplicables a los LMDB PROGRESS6 y ORACLE7. En el siguiente capítulo se analizarán los datos para contestar las preguntas e hipótesis de la investigación.

CAPITULO 5

ANALISIS DE DATOS

5.1. Introducción

En el capítulo anterior se establecieron las preguntas e hipótesis de la investigación, así como la metodología de la investigación utilizada en este estudio. En el presente capítulo se realiza el análisis estadístico de los datos con el fin de contestar las preguntas de la investigación.

El capítulo se dirige de la siguiente manera. En la sección 5.2 se discutirá la exactitud de los datos. En la sección 5.3 se presentarán las estadísticas descriptivas de los datos. En la sección 5.4 se analizarán los datos para contestar las preguntas de investigación. En la sección 5.5 se presentará un resumen del capítulo.

5.2. Exactitud de los Datos

En los programas utilizados como muestra para esta investigación se observaron dos problemas que afectan la exactitud de los datos. El primero corresponde a la forma en que están escritas las instrucciones en los programas. El segundo surge debido al número de combinaciones que se derivan de cada una de las instrucciones de los lenguajes. Estos problemas se detallan a continuación.

Debido a la flexibilidad de los lenguajes actuales que permiten a los programadores desarrollar un programa utilizando sólo las primeras letras de las instrucciones (operadores), un programa puede contener la misma instrucción escrita de diferentes formas. Esta situación puede afectar la exactitud de los datos debido a que el analizador de código puede considerar un operador como un operando. Para reducir este error se incluyó en la lista de operadores del analizador de código, las abreviaturas más comunes para las instrucciones más utilizadas en los programas. Por lo tanto la lista de operadores incluye las instrucciones escritas en forma completa, así como las abreviaciones más comunes para cada una de ellas

Aunque el número de instrucciones que contiene un lenguaje de programación es finito, el número de combinaciones que se pueden formar con los parámetros u opciones de cada instrucción es demasiado grande, por lo tanto, para cada instrucción pueden existir miles y hasta millones de combinaciones para formar una línea de instrucción de un programa. Para tratar de minimizar el problema anterior se consideró, dentro del archivo de operadores del analizador de código las combinaciones más utilizadas para cada una de las instrucciones.

Considerando los dos problemas anteriores, la lista de operadores del analizador de código contiene la mayoría de los operadores, incluyendo las combinaciones y abreviaturas más utilizadas en los programas.

Estas consideraciones se hicieron con el objetivo de no realizar modificaciones a los programas de la muestra, y así darle más veracidad a los datos obtenidos en la investigación, ya que la forma en que están escritos estos programas es similar a como se realizan en la realidad.

5.3. Estadísticas Descriptivas

En la tabla V se muestran los valores de longitud real N y de longitud estimada \hat{N} para los programas escritos en ORACLE7.

En la tabla VI se muestran los valores de longitud real N y de longitud estimada \hat{N} para los programas escritos en PROGRESS6.

Tabla V

Longitudes Reales y Longitudes Estimadas para ORACLE7

Prog	N	\hat{N}
1	7	13
2	7	13
3	7	12
4	7	12
5	9	16
6	9	16
7	9	16
8	9	16
9	9	16
10	9	20
11	9	20
12	10	20
13	10	20
14	10	18
15	11	24
16	12	23

Prog	N	\hat{N}
17	12	20
18	12	20
19	12	20
20	13	27
21	13	28
22	14	28
23	14	28
24	14	31
25	14	29
26	16	35
27	16	36
28	16	36
29	16	31
30	16	40
31	16	29
32	17	31

Prog	N	\hat{N}
33	17	29
34	18	35
35	18	40
36	20	40
37	20	48
38	20	38
39	21	44
40	21	39
41	21	44
42	22	49
43	22	48
44	22	31
45	23	48
46	24	49
47	25	57
48	25	68

Prog	N	\hat{N}
49	25	38
50	26	38
51	28	38
52	30	76
53	31	43
54	34	38
55	35	81
56	46	29
57	54	128
58	58	135
59	62	29
60	74	110
61	92	147
62	94	122
63	94	135

Tabla VI

Longitudes Reales y Longitudes Estimadas para PROGRESS6

Prog	N	N̂
1	5	7
2	8	7
3	8	13
4	8	13
5	8	13
6	8	12
7	8	14
8	8	18
9	8	12
10	8	16
11	9	16
12	9	16
13	9	16
14	10	20
15	10	20
16	10	18
17	10	16
18	10	14
19	10	16
20	10	20
21	10	20
22	10	20
23	10	18
24	10	20
25	10	20
26	11	16
27	11	20
28	11	23
29	11	24
30	11	24
31	11	23
32	11	24
33	12	28
34	12	27
35	12	24
36	12	24
37	12	27
38	12	24
39	12	24
40	12	24
41	12	24
42	12	24
43	12	28
44	12	24
45	12	24

Prog	N	N̂
46	12	24
47	12	24
48	12	20
49	12	24
50	12	28
51	12	20
52	13	27
53	13	31
54	13	31
55	14	36
56	14	27
57	14	20
58	14	31
59	14	23
60	14	27
61	14	27
62	15	20
63	15	28
64	15	36
65	15	35
66	15	24
67	15	29
68	15	31
69	16	35
70	16	40
71	16	37
72	16	31
73	16	39
74	16	40
75	16	35
76	16	40
77	16	35
78	16	18
79	16	40
80	16	35
81	16	35
82	16	18
83	16	18
84	16	31
85	16	31
86	16	31
87	17	31
88	17	40
89	17	28
90	17	28

Prog	N	N̂
91	17	36
92	17	22
93	17	31
94	17	40
95	17	44
96	17	40
97	18	44
98	18	48
99	18	32
100	18	28
101	18	24
102	19	44
103	19	40
104	19	32
105	19	32
106	19	35
107	20	44
108	20	44
109	20	46
110	20	35
111	20	40
112	20	37
113	20	32
114	20	35
115	20	53
116	20	44
117	20	35
118	20	32
119	20	40
120	20	32
121	21	20
122	21	46
123	21	40
124	21	35
125	21	44
126	21	53
127	21	36
128	21	44
129	21	26
130	21	36
131	21	48
132	22	57
133	22	62
134	22	40
135	22	44

Prog	N	N̂
136	22	41
137	22	37
138	22	44
139	23	41
140	23	35
141	24	62
142	24	62
143	24	48
144	24	53
145	24	57
146	24	68
147	24	61
148	24	37
149	24	58
150	24	58
151	25	46
152	26	67
153	26	63
154	26	44
155	26	57
156	26	64
157	26	65
158	26	58
159	26	48
160	26	60
161	26	60
162	26	63
163	27	59
164	27	53
165	27	39
166	27	44
167	28	61
168	28	35
169	28	62
170	28	62
171	28	73
172	28	67
173	28	67
174	28	72
175	29	67
176	29	55
177	29	67
178	29	57
179	29	67
180	29	41

Tabla VI (Continúa)

Prog	N	Ñ
181	30	58
182	30	76
183	30	72
184	30	67
185	30	67
186	30	50
187	30	71
188	30	71
189	30	63
190	30	88
191	30	62
192	30	54
193	31	41
194	31	53
195	31	72
196	31	50
197	31	62
198	31	54
199	31	54
200	32	91
201	32	91
202	32	91
203	32	72
204	32	68
205	32	68
206	32	68
207	32	57
208	32	49
209	32	72
210	33	67
211	33	63
212	33	76
213	33	54
214	33	86
215	33	59
216	33	54
217	33	49
218	33	57
219	33	87
220	33	58
221	33	76
222	33	54
223	34	72
224	34	63
225	34	67

Prog	N	Ñ
226	34	91
227	34	71
228	34	64
229	34	63
230	34	76
231	35	77
232	35	60
233	35	51
234	35	62
235	35	77
236	35	77
237	35	96
238	35	91
239	36	97
240	36	97
241	36	62
242	36	92
243	36	63
244	36	63
245	36	83
246	37	78
247	37	78
248	37	74
249	37	76
250	37	86
251	37	69
252	37	73
253	37	58
254	38	77
255	38	96
256	38	63
257	38	84
258	38	87
259	39	74
260	39	102
261	39	72
262	39	72
263	39	69
264	39	67
265	39	59
266	40	77
267	40	88
268	40	77
269	40	86
270	40	101

Prog	N	Ñ
271	40	91
272	41	77
273	41	91
274	41	73
275	41	93
276	41	76
277	41	72
278	41	86
279	41	88
280	42	72
281	42	86
282	42	81
283	42	91
284	42	92
285	42	77
286	42	85
287	43	104
288	43	87
289	43	67
290	43	86
291	43	84
292	43	73
293	43	80
294	43	91
295	44	77
296	44	77
297	44	62
298	44	99
299	44	92
300	44	81
301	44	81
302	44	81
303	44	77
304	44	71
305	44	88
306	44	71
307	45	81
308	45	124
309	45	93
310	45	124
311	45	68
312	45	69
313	45	95
314	45	74
315	45	91

Prog	N	Ñ
316	46	97
317	46	95
318	46	97
319	46	72
320	47	119
321	47	85
322	47	98
323	48	102
324	48	98
325	48	97
326	48	118
327	49	102
328	49	92
329	49	68
330	49	109
331	49	83
332	49	67
333	49	76
334	49	113
335	49	88
336	49	96
337	50	156
338	50	83
339	50	83
340	50	83
341	50	77
342	50	81
343	50	102
344	51	109
345	51	118
346	51	161
347	51	129
348	51	107
349	51	109
350	51	98
351	51	96
352	51	150
353	51	118
354	52	69
355	52	95
356	52	77
357	52	112
358	52	143
359	53	130
360	53	133

Tabla VI (Continúa)

Prog	N	Ñ
361	53	118
362	53	82
363	53	118
364	53	96
365	53	87
366	53	87
367	54	123
368	54	86
369	54	112
370	54	140
371	55	81
372	55	119
373	55	100
374	56	87
375	56	118
376	56	102
377	56	86
378	56	86
379	56	145
380	57	77
381	57	145
382	57	145
383	57	86
384	57	102
385	57	77
386	57	112
387	57	91
388	57	143
389	57	156
390	58	145
391	58	96
392	58	101
393	58	96
394	58	96
395	58	96
396	58	101
397	58	96
398	58	96
399	58	118
400	58	124
401	59	136
402	59	156
403	59	128
404	59	129
405	59	88

Prog	N	Ñ
406	59	93
407	59	91
408	59	112
409	59	112
410	60	108
411	60	150
412	60	93
413	60	119
414	61	134
415	61	87
416	61	123
417	61	107
418	61	117
419	62	156
420	62	139
421	62	156
422	63	102
423	63	128
424	63	161
425	63	82
426	63	145
427	64	117
428	64	168
429	64	119
430	65	168
431	65	168
432	66	169
433	66	129
434	66	129
435	67	86
436	67	134
437	68	173
438	68	113
439	68	112
440	68	78
441	68	102
442	68	91
443	68	197
444	68	139
445	69	190
446	69	128
447	69	147
448	69	81
449	69	81
450	69	163

Prog	N	Ñ
451	70	104
452	71	184
453	71	174
454	71	119
455	72	157
456	73	214
457	73	139
458	73	139
459	73	118
460	73	135
461	74	107
462	74	136
463	74	150
464	75	179
465	75	139
466	76	161
467	76	168
468	77	67
469	77	112
470	78	133
471	78	192
472	78	145
473	78	133
474	78	168
475	79	133
476	80	129
477	80	128
478	80	179
479	80	173
480	81	145
481	82	134
482	82	191
483	82	191
484	82	192
485	83	107
486	83	107
487	84	134
488	84	98
489	84	190
490	84	156
491	85	157
492	85	150
493	85	151
494	85	203
495	85	203

Prog	N	Ñ
496	86	168
497	87	141
498	87	163
499	87	139
500	87	185
501	88	238
502	88	139
503	88	191
504	88	205
505	88	128
506	89	162
507	89	174
508	89	202
509	90	246
510	90	196
511	90	220
512	91	134
513	91	157
514	92	179
515	92	215
516	93	161
517	93	123
518	94	196
519	94	196
520	94	208
521	95	244
522	96	156
523	97	185
524	97	151
525	99	282
526	99	157
527	100	263
528	100	232
529	101	238
530	102	146
531	103	251
532	103	156
533	103	168
534	103	220
535	103	220
536	104	163
537	105	173
538	105	185
539	105	185
540	106	192

Tabla VI (Continúa)

Prog	N	N̂
541	107	185
542	108	150
543	108	173
544	108	272
545	109	146
546	109	240
547	110	151
548	111	233
549	112	179
550	112	238
551	113	162
552	113	257
553	114	158
554	114	168
555	114	232
556	116	282
557	118	259
558	118	251
559	119	203
560	119	294
561	119	250
562	120	214
563	121	162
564	122	232
565	123	216
566	124	245
567	124	263
568	125	214
569	127	251
570	128	269
571	129	245
572	129	174
573	129	220
574	130	276
575	130	258
576	130	270
577	135	295
578	135	251
579	135	269
580	135	208
581	136	322
582	139	321
583	140	245
584	141	294
585	141	239

Prog	N	N̂
586	141	238
587	141	208
588	142	208
589	143	186
590	143	301
591	145	282
592	145	282
593	145	282
594	145	252
595	146	283
596	147	220
597	148	294
598	149	197
599	149	353
600	149	246
601	149	239
602	149	239
603	153	252
604	153	252
605	153	296
606	153	320
607	153	252
608	156	197
609	156	252
610	156	252
611	160	251
612	160	232
613	161	399
614	163	251
615	164	258
616	169	386
617	169	169
618	171	314
619	173	308
620	174	266
621	175	289
622	175	352
623	175	271
624	176	273
625	177	426
626	178	228
627	182	492
628	183	333
629	184	339
630	184	399

Prog	N	N̂
631	185	251
632	185	309
633	185	309
634	186	269
635	186	311
636	186	311
637	187	214
638	187	227
639	189	269
640	189	392
641	192	346
642	193	359
643	194	288
644	195	301
645	195	386
646	197	251
647	200	295
648	205	420
649	208	220
650	209	327
651	210	314
652	211	245
653	212	245
654	212	290
655	213	352
656	215	257
657	216	161
658	216	440
659	217	301
660	218	413
661	224	215
662	226	167
663	226	264
664	226	296
665	228	502
666	229	334
667	230	386
668	234	263
669	240	245
670	240	246
671	242	220
672	245	379
673	253	401
674	254	233
675	257	339

Prog	N	N̂
676	262	399
677	265	373
678	266	467
679	266	343
680	268	259
681	269	467
682	269	480
683	270	365
684	274	327
685	275	366
686	275	676
687	275	366
688	281	538
689	285	302
690	286	412
691	286	558
692	288	572
693	289	402
694	291	501
695	291	373
696	303	448
697	303	414
698	307	601
699	307	523
700	309	402
701	313	428
702	313	483
703	328	587
704	338	550
705	340	415
706	345	425
707	345	418
708	349	439
709	354	460
710	359	550
711	359	554
712	361	433
713	367	295
714	371	600
715	372	412
716	373	340
717	380	476
718	380	562
719	387	439
720	392	399

Tabla VI (Continúa)

Prog	N	Ñ
721	393	338
722	393	459
723	394	510
724	397	406
725	412	359
726	418	431
727	419	794
728	424	643
729	428	529
730	429	550
731	431	390
732	432	610
733	437	710
734	440	480
735	446	591
736	447	660
737	458	509
738	458	617
739	462	682
740	464	453
741	464	821
742	465	419
743	471	447
744	476	830
745	477	447
746	478	632
747	485	844
748	502	830
749	510	806
750	517	481
751	518	751
752	528	447
753	530	460
754	532	1016
755	537	716
756	558	732
757	566	610
758	570	756
759	586	580
760	594	922
761	597	728
762	603	675
763	610	739
764	614	544
765	614	388

Prog	N	Ñ
766	622	661
767	626	782
768	627	624
769	631	836
770	655	846
771	668	652
772	670	1050
773	672	551
774	672	551
775	675	808
776	676	689
777	684	915
778	685	611
779	694	435
780	698	858
781	719	447
782	723	581
783	730	1157
784	736	709
785	737	1111
786	743	1080
787	748	747
788	748	747
789	750	870
790	755	1055
791	756	1039
792	762	909
793	773	1302
794	782	913
795	783	1062
796	789	1140
797	793	697
798	793	697
799	793	697
800	794	860
801	804	814
802	809	1171
803	821	1021
804	824	540
805	824	395
806	824	955
807	827	461
808	828	782
809	830	919
810	838	1196

Prog	N	Ñ
811	848	1203
812	852	718
813	857	908
814	858	890
815	866	1331
816	877	901
817	879	836
818	880	742
819	886	1193
820	895	778
821	896	1193
822	899	615
823	900	904
824	916	636
825	924	847
826	937	1276
827	937	492
828	939	805
829	942	1042
830	952	1155
831	956	1063
832	965	1279
833	971	957
834	1008	922
835	1018	1043
836	1034	1058
837	1036	1314
838	1048	663
839	1057	1230
840	1064	1007
841	1088	934
842	1095	966
843	1098	769
844	1113	1297
845	1114	962
846	1121	1556
847	1126	580
848	1139	562
849	1154	833
850	1164	992
851	1204	702
852	1219	1146
853	1223	1254
854	1226	540
855	1226	533

Prog	N	Ñ
856	1236	1184
857	1244	562
858	1246	1208
859	1269	1084
860	1297	1050
861	1298	1186
862	1349	577
863	1352	1297
864	1353	1604
865	1370	898
866	1373	1506
867	1379	1532
868	1391	1037
869	1399	914
870	1402	1196
871	1404	1191
872	1408	1568
873	1443	1239
874	1456	1256
875	1472	1243
876	1481	835
877	1492	626
878	1492	610
879	1506	1695
880	1519	1328
881	1522	1238
882	1523	1247
883	1526	1287
884	1530	1281
885	1539	1602
886	1547	921
887	1571	1453
888	1577	659
889	1588	1391
890	1606	1434
891	1633	1596
892	1666	1200
893	1667	1610
894	1681	1369
895	1682	651
896	1716	1388
897	1744	1529
898	1757	1537
899	1776	1366
900	1800	1595

Tabla VI (Continúa)

Prog	N	N̂
901	1832	1426
902	1832	798
903	1839	1451
904	1853	1942
905	1861	1641
906	1874	712
907	1896	1644
908	1903	593
909	1934	1840
910	1934	1840
911	1948	1425
912	1953	1537
913	1958	640
914	1975	1604
915	1977	1493

Prog	N	N̂
916	1978	1026
917	1979	2336
918	2003	729
919	2101	778
920	2125	2163
921	2141	2152
922	2141	2152
923	2160	1608
924	2193	745
925	2194	1787
926	2207	1842
927	2207	1842
928	2208	2611
929	2217	1731
930	2269	1991

Prog	N	N̂
930	2269	1991
931	2272	2164
932	2282	1750
933	2292	862
934	2294	729
935	2334	1638
936	2345	2098
937	2397	1542
938	2525	2133
939	2571	2205
940	2840	1771
941	2840	1771
942	2876	1800
943	2980	2564
944	3078	1724

Prog	N	N̂
945	3119	2481
946	3341	1975
947	3341	1975
948	3482	1479
949	3548	2005
950	3567	1504
951	3578	2287
952	3650	1532
953	3680	1923
954	3708	1599
955	3741	1635
956	3766	1672
957	3830	1523

En la tabla VII se muestran los valores del nivel del lenguaje para los programas escritos en ORACLE7. En la tabla VIII se muestran los valores del nivel del lenguaje para los programas escritos en PROGRESS6.

Tabla VII

Nivel del Lenguaje para los Programas de ORACLE7

Prog	λ
1	4.913
2	4.913
3	2.895
4	2.895
5	4.320
6	4.320
7	4.320
8	4.320
9	4.320
10	4.565
11	2.204

Prog	λ
12	7.925
13	7.925
14	3.333
15	2.179
16	6.378
17	2.377
18	2.377
19	2.377
20	4.997
21	3.671
22	2.530

Prog	λ
23	2.530
24	4.097
25	2.297
26	3.551
27	2.570
28	9.473
29	9.178
30	3.807
31	2.625
32	2.538
33	2.789

Prog	λ
34	3.995
35	6.017
36	1.469
37	1.802
38	2.370
39	6.698
40	6.527
41	2.977
42	1.980
43	2.629
44	6.438

Prog	λ
45	2.073
46	2.160
47	4.170
48	1.958
49	2.963
50	3.081
51	3.318
52	4.560
53	3.187
54	4.029
55	3.350

Prog	λ
56	7.546
57	1.424
58	1.438
59	10.171
60	2.386
61	2.026
62	2.645
63	2.330

Tabla VIII (Continúa)

373	0.318	435	2.539	497	0.430	559	0.574	621	1.252	683	1.093
374	1.279	436	1.775	498	0.538	560	1.911	622	0.764	684	1.251
375	0.848	437	2.002	499	2.457	561	0.793	623	0.440	685	1.113
376	1.395	438	0.974	500	1.333	562	2.048	624	2.865	686	1.592
377	5.107	439	2.634	501	1.288	563	0.608	625	1.204	687	1.113
378	5.107	440	0.362	502	1.308	564	1.638	626	0.203	688	1.629
379	1.540	441	3.233	503	2.352	565	2.954	627	1.924	689	0.942
380	1.218	442	2.372	504	0.871	566	1.298	628	0.986	690	0.692
381	2.440	443	3.057	505	1.040	567	1.129	629	1.025	691	0.516
382	2.440	444	1.183	506	1.175	568	1.380	630	0.806	692	0.516
383	1.871	445	2.432	507	1.320	569	1.606	631	0.776	693	1.486
384	1.059	446	1.865	508	2.053	570	1.396	632	1.716	694	1.276
385	1.218	447	1.068	509	2.978	571	0.914	633	2.191	695	0.786
386	2.412	448	5.845	510	1.872	572	0.435	634	0.910	696	1.223
387	1.811	449	5.845	511	1.836	573	1.689	635	5.593	697	1.362
388	1.027	450	0.932	512	2.470	574	1.023	636	5.593	698	1.729
389	1.851	451	0.688	513	0.896	575	1.033	637	0.586	699	0.404
390	2.482	452	1.702	514	1.147	576	1.168	638	1.440	700	2.125
391	5.564	453	1.388	515	1.659	577	1.329	639	0.978	701	1.258
392	6.527	454	1.059	516	0.988	578	1.103	640	2.958	702	1.452
393	5.564	455	1.261	517	1.319	579	1.285	641	6.742	703	0.984
394	5.564	456	2.045	518	1.671	580	1.217	642	1.054	704	0.611
395	5.564	457	1.651	519	1.671	581	1.814	643	0.969	705	1.423
396	2.157	458	1.651	520	1.131	582	1.862	644	0.813	706	0.201
397	2.003	459	3.625	521	2.115	583	0.715	645	0.844	707	0.208
398	5.564	460	1.976	522	1.066	584	1.143	646	1.007	708	0.616
399	1.045	461	2.105	523	1.953	585	2.315	647	0.989	709	0.576
400	0.963	462	0.706	524	0.885	586	1.280	648	1.378	710	5.700
401	1.761	463	1.700	525	1.559	587	0.954	649	0.507	711	0.949
402	2.026	464	2.563	526	0.631	588	1.234	650	0.775	712	0.440
403	1.475	465	1.696	527	1.722	589	0.410	651	0.951	713	3.695
404	5.244	466	1.773	528	1.484	590	1.793	652	0.928	714	0.419
405	0.613	467	1.470	529	2.238	591	1.123	653	0.933	715	0.449
406	0.717	468	0.870	530	2.432	592	1.123	654	0.341	716	0.508
407	1.398	469	2.356	531	1.303	593	1.123	655	0.862	717	0.255
408	1.598	470	1.778	532	1.297	594	0.949	656	0.875	718	0.226
409	2.064	471	3.636	533	1.253	595	2.205	657	0.658	719	0.676
410	0.998	472	1.440	534	1.831	596	1.463	658	0.750	720	0.377
411	2.659	473	2.266	535	1.831	597	1.245	659	1.206	721	1.448
412	0.729	474	1.181	536	0.726	598	0.559	660	0.716	722	1.524
413	0.895	475	1.679	537	0.890	599	1.155	661	0.397	723	1.831
414	2.137	476	0.790	538	2.788	600	0.531	662	0.458	724	0.520
415	1.294	477	1.432	539	2.788	601	1.795	663	0.378	725	4.829
416	1.605	478	2.245	540	0.660	602	1.795	664	0.344	726	0.190
417	2.660	479	2.077	541	1.041	603	0.465	665	0.579	727	1.193
418	2.079	480	1.647	542	1.460	604	2.301	666	1.408	728	0.647
419	1.494	481	2.327	543	1.243	605	0.793	667	0.959	729	0.528
420	2.385	482	1.365	544	2.304	606	1.154	668	0.650	730	0.473
421	2.645	483	1.365	545	2.278	607	2.301	669	0.598	731	0.189
422	1.227	484	3.212	546	1.071	608	0.529	670	1.409	732	0.951
423	1.451	485	0.877	547	2.543	609	2.346	671	0.403	733	0.777
424	1.860	486	0.877	548	0.877	610	2.346	672	0.866	734	1.522
425	4.789	487	0.222	549	2.116	611	1.321	673	1.476	735	1.180
426	1.262	488	6.700	550	1.255	612	0.855	674	1.256	736	0.905
427	1.987	489	1.968	551	1.727	613	1.380	675	0.704	737	1.706
428	1.527	490	1.558	552	0.716	614	0.956	676	0.847	738	0.190
429	1.166	491	0.603	553	0.360	615	10.021	677	0.654	739	0.946
430	1.551	492	1.829	554	2.288	616	0.718	678	0.649	740	0.566
431	1.551	493	0.908	555	1.305	617	0.311	679	1.717	741	0.470
432	2.474	494	1.806	556	1.950	618	1.133	680	1.464	742	0.471
433	3.657	495	1.806	557	0.813	619	0.755	681	0.770	743	0.597
434	2.746	496	1.302	558	0.768	620	2.745	682	0.761	744	0.825

Tabla VIII (Continúa)

Prog	λ	Prog	λ	Prog	λ	Prog	λ	Prog	λ	Prog	λ
745	0.583	782	0.168	819	0.376	856	0.437	893	0.715	930	0.340
746	0.989	783	0.564	820	0.171	857	3.803	894	1.755	931	3.055
747	1.384	784	0.350	821	0.300	858	0.859	895	2.825	932	0.304
748	0.663	785	0.974	822	0.186	859	1.443	896	0.243	933	3.517
749	0.580	786	0.615	823	0.306	860	0.501	897	0.279	934	3.397
750	0.533	787	0.336	824	0.203	861	0.852	898	0.272	935	0.229
751	0.967	788	0.336	825	0.669	862	3.696	899	0.296	936	1.299
752	0.597	789	0.719	826	0.977	863	0.590	900	0.275	937	0.807
753	0.588	790	0.702	827	3.912	864	0.655	901	0.729	938	0.342
754	4.212	791	0.500	828	0.279	865	0.226	902	0.169	939	0.361
755	0.974	792	0.778	829	0.621	866	0.371	903	0.336	940	11.896
756	0.652	793	2.787	830	0.404	867	0.398	904	0.974	941	11.896
757	0.572	794	0.736	831	0.500	868	0.217	905	0.757	942	9.343
758	0.797	795	0.595	832	0.731	869	0.221	906	3.988	943	0.368
759	0.263	796	0.744	833	0.309	870	0.659	907	0.628	944	0.357
760	0.777	797	0.518	834	0.459	871	0.601	908	2.767	945	0.302
761	0.718	798	0.518	835	0.287	872	0.252	909	2.599	946	12.327
762	0.773	799	0.518	836	0.515	873	1.659	910	2.599	947	12.327
763	0.417	800	0.835	837	0.346	874	1.691	911	0.386	948	0.447
764	0.463	801	0.355	838	0.508	875	0.344	912	0.584	949	9.226
765	3.754	802	0.731	839	0.611	876	0.185	913	3.133	950	0.435
766	1.491	803	1.175	840	0.431	877	1.629	914	0.465	951	0.189
767	0.696	804	1.381	841	1.679	878	1.546	915	0.538	952	0.458
768	1.364	805	2.886	842	0.241	879	0.755	916	0.394	953	0.340
769	0.448	806	0.678	843	0.245	880	1.543	917	0.915	954	0.463
770	0.664	807	0.266	844	0.277	881	1.187	918	3.887	955	0.483
771	0.441	808	1.397	845	0.657	882	1.202	919	4.222	956	0.521
772	0.686	809	0.275	846	1.136	883	1.580	920	0.563	957	0.431
773	0.440	810	1.130	847	3.362	884	0.417	921	0.525		
774	0.440	811	1.085	848	4.159	885	0.383	922	0.525		
775	0.525	812	0.192	849	0.370	886	0.189	923	0.140		
776	0.599	813	0.875	850	1.218	887	0.279	924	3.715		
777	0.608	814	0.371	851	0.200	888	3.052	925	0.238		
778	0.778	815	0.968	852	0.789	889	0.591	926	1.062		
779	1.673	816	0.597	853	0.580	890	0.599	927	1.062		
780	0.394	817	0.356	854	1.471	891	0.610	928	0.653		
781	4.240	818	0.650	855	1.428	892	0.336	929	0.767		

En la tabla IX se muestra la media y desviación estándar del nivel del lenguaje para PROGRESS6 y ORACLE7.

En la tabla X se muestran las distribuciones de frecuencia de los niveles del lenguaje para los programas del lenguaje ORACLE7. En la tabla XI se muestran las distribuciones de frecuencia de los niveles del lenguaje para los programas del lenguaje PROGRESS6.

Tabla IX

Nivel del lenguaje Promedio para PROGRESS Y ORACLE

Lenguaje	λ	Desviación Estandar
PROGRESS6	1.9734	1.8295
ORACLE7	3.8607	2.0579

Tabla X

Distribuciones de Frecuencia de Niveles del lenguaje para ORACLE7

Intervalo	Frecuencia	Porcentaje	Porcentaje Acumulado
[0, 1)	6	9.50	9.50
[1, 2)	24	38.10	47.60
[2, 3)	9	14.30	61.90
[3, 4)	13	20.60	82.50
[4, 5)	0	0.00	82.50
[5, 6)	5	7.90	90.50
[6, 7)	3	4.80	95.20
[7, 8)	0	0.00	95.20
[8, 9)	2	3.20	98.40
[9, 10)	1	1.60	100.00

Tabla XI

Distribuciones de Frecuencia de Niveles del lenguaje para PROGRESS6

Intervalo	Frecuencia	Porcentaje	Porcentaje Acumulado
[0, 1)	340	35.50	35.50
[1, 2)	301	31.50	67.00
[2, 3)	145	15.20	82.10
[3, 4)	53	5.50	87.70
[4, 5)	37	3.90	91.50
[5, 6)	42	4.40	95.90
[6, 7)	15	1.60	97.50
[7, 8)	10	1.00	98.50
[8, 9)	4	0.40	99.00
[9, 10)	5	0.50	99.50
[10, 11)	1	0.10	99.60
[11, 12)	2	0.20	99.80
[12, 13)	2	0.20	100.00

Para este estudio se clasificaron los programas de acuerdo a su tamaño y función. Se utilizaron tres rangos de tamaño de programa los cuales son: menores de 500, de 500 a 1500, y mayores de 1500 partículas elementales (operadores y operandos). La clasificación para la función de los programas es la siguiente:

- *Actualizaciones*: Programas que realizan modificaciones a los datos almacenados en la base de datos.
- *Consultas*: Lista de datos almacenados en la base de datos; en esta clase de programa el usuario especifica qué información desea consultar.

- *Administración de la Base de Datos*: Programas que son utilizados por el administrador de la base de datos para mejorar el desempeño y utilización de la base de datos.
- *Funciones Matemáticas*: Programas que realizan funciones matemáticas que no están incluidas en el LMBD.
- *Menú*: Programas que presentan un conjunto de opciones, de las cuales el usuario selecciona una de ellas. Por lo general se utilizan para enlazar varios programas.
- *Reportes*: Lista que siempre presenta los datos con el mismo formato.
- *Otros*: Programas de utilidad como editores, compiladores, generadores de menús y reportes, juegos, gráficas, indexación, arreglos dimensionados, entre otros.

Esta clasificación sólo se realizó para el lenguaje PROGRESS6, debido a que todos los programas de ORACLE7 analizados en este estudio eran programas clasificados como consultas y de tamaño menor a 500 partículas elementales.

La tabla XII muestra el nivel del lenguaje para cada una de las clasificaciones por tamaño y función para PROGRESS6.

La tabla XIII muestra la desviación estándar de los niveles del lenguajes mostrados en la tabla XII.

Tabla XII

**Nivel del lenguaje para Programas de PROGRESS6
Clasificados por Función y Tamaño**

Clasificación	< 500	500-1500	> 1500	Todas
Actualizaciones	2.2888	-	-	2.2888
Consultas	1.6980	-	-	1.6980
Admón. de B.D.	1.8318	0.9851	1.9915	1.6281
F. Matemáticas	0.8397	-	-	0.8397
Menús	1.6164	0.3895	1.4389	1.3901
Reportes	2.8823	-	-	2.8823
Otros	2.0186	0.7289	0.6257	1.6279
Todas	2.1797	0.8870	1.8248	1.9734

Tabla XIII

**Desviación Estándar para Programas de PROGRESS6
Clasificados por Función y Tamaño**

Clasificación	< 500	500-1500	> 1500	Todas
Actualizaciones	1.5402	-	-	1.5402
Consultas	0.8583	-	-	0.8583
Admón. de B.D.	1.7066	0.9483	3.2047	1.9801
F. Matemáticas	0.2587	-	-	0.2587
Menús	0.6949	0.2785	1.0514	0.8440
Reportes	1.9077	-	-	1.9077
Otros	2.0023	0.7992	0.2658	1.8224
Todas	1.7325	0.8976	2.9264	1.8296

5.4. Análisis de Datos para las Preguntas de la Investigación

La primer pregunta de la investigación se estableció como: ¿Es el estimador de un programa \hat{N} un buen estimador de N , en los lenguajes manipuladores de bases de datos?. Por lo tanto, la primer hipótesis es:

H_1 : Para programas en lenguajes manipuladores de bases de datos,

$$\hat{N} = N.$$

Para contestar a esta pregunta se realizó la prueba de correlación de Pearson entre la longitud real \hat{N} y la longitud estimada N para los programas de los lenguajes PROGRESS6 y ORACLE7.

El coeficiente de correlación de Pearson (r) provee una medida cuantitativa de la fuerza de la relación lineal entre dos variables [McClave et al, 1991]. El valor de r siempre está entre -1 y $+1$ sin importar cuales sean las unidades de medición de las variables. Un valor de r cercano ó igual a cero implica que hay poca o no hay relación lineal entre las variables. Si el valor de r es cercano a $+1$ ó -1 , existe una fuerte relación entre las variables. Si $r = 1$ ó $r = -1$ entonces existe una perfecta relación entre las variables. La Tabla XIV muestra los resultados de la prueba de correlación de Pearson para PROGRESS6 y ORACLE7.

Tabla XIV

**Coeficiente de Correlación de Pearson
entre Longitud Real y Longitud Estimada**

Lenguaje	r
PROGRESS6	0.9100
ORACLE7	0.8801

Los resultados de la tabla XIV indican que N y \hat{N} están fuertemente correlacionados, por lo tanto podemos establecer que el estimador de la longitud propuesto por Halstead es un buen estimador de la longitud de los programas escritos en los LMBD PROGRESS6 y ORACLE7

La hipótesis, correspondiente a la segunda pregunta de investigación, es:

H_2 : Para programas en lenguajes manipuladores de bases de datos con diferente de tamaño y función, $\hat{N} = N$.

Para contestar a esta pregunta se calculó el coeficiente de correlación de Pearson para cada clase de programas de acuerdo a su función y su tamaño. La tabla XV muestra los resultados de aplicar el coeficiente de correlación de Pearson a los programas clasificados por tamaño y función.

Tabla XV

Coeficiente de Correlación de Pearson entre Longitud Real y Longitud Estimada para Programas de PROGRESS6 Clasificados por Tamaño y Función

Clasificación	< 500	500-1500	> 1500	Todas
Actualizaciones	0.9002	-	-	0.9002
Consultas	0.9663	-	-	0.9663
Admón. de B.D.	0.9003	0.4796	0.4307	0.8681
F. Matemáticas	0.9842	-	-	0.9842
Menús	0.9521	0.0749	0.5204	0.9556
Reportes	0.9241	-	-	0.9241
Otros	0.9359	0.4014	0.3286	0.9387
Todas	0.9348	0.4427	0.3831	0.9100

Los resultados de la tabla XV indican que el estimador de la longitud \hat{N} tiene diferente comportamiento según el tamaño y la función de los programas. Considerando el tamaño del programa, observamos que se realizan buenas estimaciones para programas de longitud menor a 500, y para programas de longitud mayor a 500 se realizan malas estimaciones. De acuerdo a la clasificación por función, las estimaciones fueron buenas para todas las clasificaciones siendo las mejores estimaciones para programas que realizan funciones matemáticas, consultas y menús.

La tercer pregunta de la investigación se estableció como: ¿Es el nivel del lenguaje de los LMBD mayor que el nivel del lenguaje de los 3GL's y menor que el nivel del lenguaje del Inglés Prosaico?. Para contestar a esta pregunta se realizaron pruebas Z entre las medias de los niveles del lenguajes de los LMBD seleccionados.

La prueba Z se utiliza para probar hipótesis sobre la media de una población en muestras grandes y para probar hipótesis sobre la diferencia entre las medias de dos poblaciones con muestras independientes, entre otras [McClave et al, 1991].

Para esta investigación se tomó como nivel del lenguaje para los 3GL's el valor de 1.53, el cual corresponde al lenguaje PL/I, que es el lenguaje que tiene el mayor nivel del lenguaje de los 3GL's. Igual que en el estudio realizado por Halstead, para el Inglés Prosaico se utilizó como nivel del lenguaje el valor de 2.16. [Halstead, 1977].

El primer par de hipótesis correspondiente a la segunda pregunta de investigación es:

H_0 : Para lenguajes manipuladores de bases de datos, el nivel del lenguaje es menor o igual a 1.53

H_a : Para lenguajes manipuladores de bases de datos, el nivel del lenguaje es mayor a 1.53

La Tabla XVI muestra los resultados de la prueba Z para el par de hipótesis anteriores. Estos resultados indican que para los LMBD PROGRESS6 y ORACLE7, hay suficiente evidencia estadística para concluir que la media del nivel del lenguaje es mayor a 1.53.

Tabla XVI

Prueba Z entre LMBD y 3GL's

Lenguaje	Z	Nivel de significancia
PROGRESS6	7.5000	0.0000
ORACLE7	8.9900	0.0000

El segundo par de hipótesis correspondiente a la segunda pregunta de la investigación es:

H_0 : Para lenguajes manipuladores de bases de datos, el nivel del lenguaje es mayor o igual a 2.16

H_a : Para lenguajes manipuladores de bases de datos, el nivel del lenguaje es menor a 2.16

La Tabla XVII muestra los resultados de la prueba Z para el par de hipótesis anteriores.

Tabla XVII

Prueba Z entre LMBD y el Inglés Prosaico

Lenguaje	Z	Nivel de significancia
PROGRESS6	-3.1545	0.0008
ORACLE7	6.5600	0.0000

Los resultados de la tabla XVII indican que para el LMBD PROGRESS6 existe suficiente evidencia para concluir que la media del nivel del lenguaje es menor a 2.16. Para el LMBD ORACLE7 no hay suficiente evidencia estadística para concluir que la media del nivel del lenguaje es menor a 2.16.

La siguiente hipótesis correspondiente a la tercera pregunta de investigación es:

H₃: Para programas en lenguajes manipuladores de bases de datos, el nivel de lenguaje es diferente según el tamaño y la función que realizan.

Los resultados de las tablas XII y XIII indican que el nivel del lenguaje promedio varía según la función y el tamaño del programa. El nivel del lenguaje es alto para programas pequeños y para programas que realizan actualizaciones y reportes. Por otra parte el nivel del lenguaje es bajo para programas grandes y para programas que realizan funciones matemáticas.

5.5. Resumen

En este capítulo se realizó el análisis estadístico de los datos para responder a las preguntas de la investigación. Los resultados son los siguientes:

1. Existe suficiente evidencia estadística para establecer que el estimador de la longitud de un programa es buen estimador para los programas escritos en los LMBD PROGRESS6 y ORACLE7.
2. Existe suficiente evidencia estadística para concluir que para el LMBD PROGRESS6 la estimación de la longitud de un programa utilizando la métrica de longitud de un programa propuesta por Halstead varía según su función y su tamaño, siendo las mejores estimaciones para programas pequeños y para programas que realizan funciones matemáticas, consultas y menús.
3. Existe suficiente evidencia estadística para concluir que el nivel del lenguaje del LMBD PROGRESS6 es mayor que el nivel del lenguaje de los lenguajes de tercera generación y menor que el nivel del lenguaje del Inglés Prosaico.
4. Existe suficiente evidencia estadística para concluir que el nivel del lenguaje del LMBD ORACLE7 es mayor que el nivel del lenguaje de los lenguajes de tercera generación, pero no existe evidencia estadística para asegurar que el nivel del lenguaje de ORACLE7 es menor que el nivel del lenguaje del Inglés Prosaico.
5. Existe suficiente evidencia estadística para concluir que para el LMBD PROGRESS6 el nivel del lenguaje de un programa varía según su función y su tamaño. El nivel del lenguaje es alto para programas pequeños y para programas que realizan actualizaciones y reportes. Por otra parte, el nivel es bajo para programas grandes y para programas que realizan funciones matemáticas.

CAPITULO 6

CONCLUSIONES

6.1. Introducción

En este capítulo se discutirán los resultados de los datos analizados en el capítulo anterior. En la sección 6.2 se recordarán los objetivos de este estudio. En la sección 6.3 se presentarán las discusiones y conclusiones de los objetivos mencionados en la sección anterior. En la sección 6.4 se presentan recomendaciones para estudios futuros.

6.2. Objetivo del Estudio

En este estudio se aplicaron las métricas de longitud del programa y nivel del lenguaje propuestas en la Ciencia del Software de Halstead a los LMBD PROGRESS6 y ORACLE7 con el objetivo de contestar a las siguientes preguntas de investigación:

1. ¿Es la métrica de longitud del programa propuesta por Halstead un buen estimador para la longitud de los programas escritos en los LMBD PROGRESS6 y ORACLE7?
2. ¿Afecta el tamaño y la función de un programa en la estimación de la longitud de programas escritos en los LMBD PROGRESS6 y ORACLE7?

3. Es el nivel del lenguaje de los LMBD PROGRESS6 Y ORACLE7 mayor que los 3GL's y menor que el Inglés Prosaico?

6.3. Discusiones y Conclusiones

6.3.1. Discusión y Conclusión de la Primer Pregunta de Investigación

Según el análisis estadístico de los datos, existe una fuerte correlación entre el estimador de la longitud propuesto por Halstead y la longitud real de programas escritos en los LMBD PROGRESS6 y ORACLE7.

Para PROGRESS este resultado sólo es válido para programas cuya longitud esté entre 5 y 3830 partículas elementales, mientras que para ORACLE7 este resultado sólo es válido para programas con longitudes entre 7 y 94 partículas elementales.

En este estudio se observó, al igual que en estudios anteriores [Shen et al, 1983; Martínez, 1994], que el estimador de la longitud de un programa comienza sobrestimando la longitud de un programa y termina subestimándola (ver figura 9). Este resultado puede verse en la tablas VI. Estos resultados se observaron para los programas del LMBD PROGRESS6 debido a que la muestra contenía programas de diferentes tamaños.

Para el LMBD ORACLE7, aunque se encontró que existe una fuerte correlación entre la longitud de un programa y su estimador, sólo se observó que el estimador comienza sobrestimando la longitud, debido a que la muestra de este LMBD contenía sólo programas pequeños.

Por lo tanto podemos concluir que para los LMBD PROGRESS6 y ORACLE7 el estimador de la longitud propuesto por Halstead es un buen estimador de la longitud.

6.3.2. Discusión y Conclusión de la Segunda Pregunta de Investigación.

El segundo objetivo de esta investigación es determinar si el tamaño y la función de un programa afectan la estimación de la longitud de programas escritos en los LMBD PROGRESS6 y ORACLE7.

Los programas del LMBD PROGRESS6 se dividieron en tres clases según el tamaño (menores de 500, 501-1500, mayores a 1500 partículas elementales) y en siete clases según su función (actualizaciones, consultas, administración de la base de datos, funciones matemáticas, menús, reportes y otros). Se observó que las mejores estimaciones se realizan para los programas pequeños (menores a 500 partículas elementales) y para programas que realizan funciones matemáticas, consultas y menús.

Para el LMBD ORACLE7 no fue posible clasificar los programas, ni por tamaño, ni por función debido a que todos los programas considerados para este estudio eran programas muy pequeños (menores a 100 partículas elementales) y realizaban la misma función (consultas). Por lo tanto no se pueden concluir que para el LMBD ORACLE7 el tamaño y la función de un programa afecta a la estimación de la longitud de un programa.

En base a estos resultados podemos concluir que para el LMBD PROGRESS6, el tamaño y la función de un programa afecta la estimación de su longitud utilizando la métrica de longitud de programa propuesta por Halstead, obteniéndose las mejores estimaciones para programas pequeños (menores a 500 partículas elementales) y para programas que realizan funciones matemáticas, consultas y menús. Sin embargo, este

resultado no se observó en el LMBD ORACLE7 debido a que todo los programas tienen la misma clasificación tanto en tamaño como en función.

6.3.3. Discusión y Conclusión de la Tercer Pregunta de Investigación.

La última pregunta de investigación de este estudio es determinar si el nivel del lenguaje de los LMBD PROGRESS6 Y ORACLE7 es mayor que los 3GL's y menor que el Inglés Prosaico.

Al igual que en estudios anteriores [Shen et al, 1983; Martínez, 1994], en este estudio se observó que el nivel del lenguaje depende del tamaño del programa como una función exponencial decreciente (ver figura 10). Este resultado puede ser observado para el LMBD PROGRESS6 en la tabla VIII.

Para el LMBD PROGRESS6 existe suficiente evidencia estadística para concluir que su nivel del lenguaje (1.97) es mayor que el nivel del lenguaje de los 3GL's (1.53) y menor que el Inglés Prosaico (2.16). Este resultado es lógico ya que los LMBD son considerados como 4GL's, y además es similar al nivel del lenguaje obtenido anteriormente para los 4GL's FOXPRO2 y DBASEIII, 1.97 y 1.95 respectivamente [Martínez, 1994]. Debido al nivel del lenguaje obtenido, podemos establecer que realizar programas en PROGRESS6 es más fácil que realizarlos en un 3GL's y es casi como escribir en el lenguaje natural.

Al observar los resultados de la tabla XII, donde se muestra el nivel del lenguaje para los programas de PROGRESS6 clasificados por función y tamaño, podemos observar que los programas tienen diferente nivel del lenguaje según su función y tamaño. Los programas clasificados como menores a 500 partículas elementales, tienen un alto nivel (2.17) y los programas entre 500 y 1500 partículas tiene un nivel del

lenguaje muy bajo (0.88), lo cual confirma que el estimador del nivel del lenguaje es alto para programas pequeños y bajo para programas grandes, es decir, el nivel del lenguaje depende del tamaño del programa. Sin embargo, en los programas mayores a 1500 partículas se observó que el nivel del lenguaje fue alto (1.82), esto tal vez se debe a que la muestra de programas para esta clasificación son pocos (79) comparados con la muestra total (957).

La desviación estándar de los programas de PROGRESS6 clasificados por función y tamaño, contenidos en la tabla XIII, muestra que para programas clasificados como menores a 500 partículas elementales la desviación es alta (1.73), mientras que para programas entre 500 y 1500 partículas la desviación no es tan alta (0.89). Esto puede explicarse también con la característica del estimador del nivel del lenguaje antes observada, donde sobrestima el nivel del lenguaje de los programas pequeños, mientras que para programas grandes subestima el nivel. Por lo tanto, si el nivel del lenguaje es alto para programas pequeños mayor es la desviación estándar, a diferencia de los programas grandes donde el nivel del lenguaje es pequeño, al igual que su desviación estándar.

Otro factor que interviene para que la desviación estándar sea alta, con respecto a la media del nivel del lenguaje de PROGRESS6, es el hecho que los programas tienen diferente nivel del lenguaje según la función que realizan. Como puede observarse en la tabla XII y XIII los programas tienen diferente nivel del lenguaje y la desviación estándar es pequeña en la mayoría de los casos, cuando los programas son divididos en clases según su función. El nivel del lenguaje es alto para programas pequeños y para programas que realizan actualizaciones y reportes. Por otra parte, el nivel es bajo para programas grandes y para programas que realizan funciones matemáticas.

Para el LMBD ORACLE7 existe suficiente evidencia estadística para concluir que su nivel del lenguaje (3.86) es mayor que el nivel del lenguaje de los 3GL's, pero no existe suficiente evidencia para concluir que sea menor que el nivel del lenguaje del Inglés Prosaico. Este valor tan alto puede explicarse debido a que los programas considerados en este estudio son programas demasiados pequeños (menores de 100 partículas elementales). Debido a que el estimador del nivel del lenguaje propuesto por Halstead depende del tamaño del programa, para programas pequeños el nivel del lenguaje es alto y para programas grandes el nivel es bajo. Por lo tanto como los programas del lenguaje ORACLE7 son programas muy pequeños el nivel del lenguaje promedio es muy alto.

Considerando los resultados anteriores podemos concluir que la métrica de nivel del lenguaje es válida para el LMBD PROGRESS6. También puede concluirse que el nivel del lenguaje depende del tamaño del programa que se está analizando y es diferente para cada programa según la función que realice. Para el LMBD ORACLE7 la métrica de nivel del lenguaje no es aplicable.

6.4. Recomendaciones

En esta tesis se realizó un estudio sobre las Métricas de Halstead en los lenguajes manipuladores de bases de datos PROGRESS6 y ORACLE7. Los resultados de este estudio pueden ser utilizados (en ambientes de desarrollo que utilicen los LMBD analizados) para:

1. Medir el desempeño de un grupo de programadores en alguna organización, y así poder determinar cuál es el mejor programador. Esto se puede lograr utilizando el analizador de código para evaluar un conjunto de programas desarrollados por cada

uno de los programadores, los cuales tendrán un nivel del lenguaje diferente debido a sus características individuales. Entre más alto sea este valor los programas desarrollados por los programadores son más fáciles de entender, mantener, actualizar y depurar.

2. Establecer niveles del lenguaje para el desarrollo de programas. Esto puede lograrse mediante la evaluación de los diferentes tipos de programas (reportes, consultas, menús, actualizaciones, etc.) que existen en la organización y así establecer un nivel del lenguaje mínimo, de esta manera los nuevos desarrollos deben ajustarse a los niveles obtenidos previamente. Por lo tanto, los programadores tendrán que ajustarse a los estándares de programación, lo cual mejorará la calidad de los programas.
3. Evaluar la calidad de programas existentes. Esto se puede lograr utilizando el analizador de código para determinar el nivel del lenguaje de los programas existentes en una organización y así determinar la calidad de los mismos, entre más alto sea el valor del nivel del lenguaje, mayor será la calidad de los programas.
4. Planear nuevos proyectos. Al utilizar las métricas de la longitud y de nivel del lenguaje, un programa una organización mantendrá información histórica sobre los programas desarrollados y utilizados en una empresa. Para estimar el tamaño y la calidad de nuevos desarrollos se pueden utilizar los datos históricos de programas similares.

Por último, como sugerencia para estudios futuros sobre las Métricas de Halstead, se recomienda utilizar la metodología empleada en este estudio en:

- LMBD no incluidos en este estudio como: ACCESS, SYBASE, INFORMIX, etc.
- Lenguajes de Inteligencia Artificial tales como: PROLOG, LISP, VPX.
- Lenguajes de Cuarta Generación “Visuales” como: VISUAL BASIC, VISUAL FOXPRO, VISUAL C, VISUAL DBASE, etc.

REFERENCIAS

- Athey, Thomas H y Zmud, Robert W, Introduction to Computers and Information Systems, Second Edition, Scott, Foresman and Company, 1988.
- Bowman, Brent J. y Newman, William A, “Software Metrics as a Programming Training Tool”, J. Systems Software, Vol. 13, 1990.
- Briand, Lionel C., Morasca, Sandro y Basili, Victor R. “Property-Based Measurement Engineering Software”, IEEE Transactions on Software Engineering, Vol. 22, No. 1, 1996.
- Courtney, Richard E. y Gustafson, David A. “Shotgun Correlations in Software Measures”, Software Engineering Journal, 1993.
- Fenton, Norman. “Software Measurement: A Necessary Scientific Basis”, IEEE Transactions on Software Engineering, Vol. 20, No. 3, 1994.
- Halstead, Maurice H. Elements of Software Science. North Holland New York. 1977.
- Henry, Sallie y Kafura, Denis. “The Evaluation of Software Systems’ Structure Using Quantitative Software Metrics”, Software -Practice and Experience, Vol. 14(6), 1984.
- Hernández Sampieri, R., Fernández Collado, C. y Baptista Lucio, P., Metodología de la Investigación, McGraw-Hill. 1991.

INEGI, Situación de la Informática en México, Instituto Nacional de Estadística Geografía e Informática, 1992.

Johnson, G. Vaughn, Information Systems: A Strategic Approach, Mountain Top Publishing, 1990.

Keller-McNulty, S., McNulty, Mark S. y Gustafson, David A., “Stochastic Models for Software Science”, J. Systems Software, Vol 16, 1991.

Kruglinski, David, Sistemas de Administración de Base de Datos, McGraw-Hill / A Osborne, 1985.

Lokan, Christopher J. “Early Size Prediction for C and Pascal Programs”, J. Systems Software, Vol 32, 1996.

Martínez Flores, José Luis, “Métricas de Software en Lenguajes de Cuarta Generación”, Tesis de Maestría, Facultad de Ingeniería Mecánica y Eléctrica, U.A.N.L., 1994.

McClave, James T. y Benson, P. George, Statistics for Business and Economics, Maxwell Macmillan International, Fifth Edition, 1991.

Möller, K. H. y Paulish, D. J., Software Metrics A Practitioner's Guide to Improved Product Development, Champan & Hall Computing, 1993.

O'brien, James A. Management Information Systems. IRWIN. 1990.

Pressman, Roger S., Ingeniería del Software: un enfoque práctico, Tercera Edición, McGraw-Hill, 1993.

- Senn, James A, Análisis y Diseño de Sistemas de Información, Segunda Edición, McGraw-Hill, 1992
- Shen, Vincent. Y., Conte, Samuel D. y Dunsmore, H. E., “Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support”, IEEE Transactions on Software Engineering, Vol. 9, No. 2, 1983.
- Tian, Jeff y Zelkowitz, Marvin V, “Complexity Measure Evaluation and Seleccion”. IEEE Transactions on Software Engineering, Vol. 21, No. 8, 1995.
- VARB, “PROGRESS Outshines Competitors As Top Sql Database/Development Tool”, VARBusiness Magazine, <http://www.progress.com/News/varbus3.html>, 1995.
- Verner, J. y Tate, G. “A Software Size Model”. IEEE Transactions on Software Engineering, Vol. 18, No.4, 1992.
- Weyuker, J. Elaine. “Evaluating Software Complexity Measures”. IEEE Transactions on Software Engineering, Vol. 14, No. 9, 1988.
- Winblad, Ann L., Edwards, Samuel D. y King, David R, Software orientado a objetos, Addison-Wesley / Diaz de Santos, 1993.
- Wrigley, Clive D. y Dexter, Albert S., “A Model for Measuring Information Systems Size”, MIS Quarterly, 1991.

APENDICE

ANALIZADOR DE CODIGO

APENDICE

ANALIZADOR DE CODIGO

El analizador de código utilizado en este estudio fue desarrollado inicialmente para aplicar las Métricas de Halstead a los 4GL's FOXPRO2 y DBASEIII [Martínez, 1994]. Este analizador está implementado en el lenguaje PASCAL versión 6.0 y gracias a su diseño, permite realizar modificaciones de tal forma que pueda ser empleado en otros lenguajes de programación.

El analizador de código se utilizó en este estudio con los siguientes objetivos:

1. Evitar errores humanos en la interpretación de las partes que componen el programa. Esto es, si una palabra es operador o no.
2. Evitar el conteo manual de los diferentes operadores y operandos que forman parte del código fuente de un programa; es posible olvidar si una palabra es operador u operando.
3. Obtener las Métricas de Halstead para cualquier programa fuente.

Para utilizar el analizador de código diseñado por Martínez [1994] en los LMBD PROGRESS6 y ORACLE7 se realizaron las siguientes modificaciones:

1. Crear un archivo de operadores y un archivo de basura para el lenguaje PROGRESS6.
2. Crear un archivo de operadores y un archivo de basura para el lenguaje ORACLE7.

3. Modificar el código fuente del analizador para eliminar: a) los comentarios; b) líneas en blanco; y c) espacios en blanco entre operadores y operandos cuando eran más de uno. Esto se hizo debido a que los comentarios en los programas de PROGRESS6 pueden abarcar de una a varias líneas con un mismo identificador de comentarios, a diferencia de FOXPRO2 y DBASEIII en los que se emplea un identificador por cada línea de comentarios.
4. Modificar el código fuente para eliminar los operadores “basura” (operadores que en su sintaxis emplean una instrucción para indicar el fin de la instrucción), principalmente el operador “END”, ya que puede ser utilizado en diferentes formas para indicar la terminación de un conjunto de instrucciones.

En las figuras 11 y 12 se muestran los diagramas de flujo de datos del analizador de código. A continuación se describen las entidades, archivos, procesos y flujos de datos utilizados en los diagramas:

1. *Basura*: Archivo que contiene las palabras que no tienen efecto en el conteo.
2. *Operadores*: Archivo que contiene la lista de los operadores del lenguaje analizado.
3. *Operandos*: Archivo que almacena los operandos encontrados durante el análisis de un programa.
4. *Limpiar basura*: Proceso que elimina la basura de un programa.
5. *Borra comentarios*: Proceso que elimina comentarios, espacios en blanco, tabulaciones y líneas en blanco.
6. *Comillas*: Proceso que elimina los operandos que viene entre comillas dentro de un programa.
7. *Modificación 1*: Programa sin basura.
8. *Modificación 2*: Programa sin comentarios, espacios en blanco, tabulaciones y líneas en blanco.
9. *Modificación 3*: Programa sin comillas.
10. *Modificación 4*: Programa sin operadores.

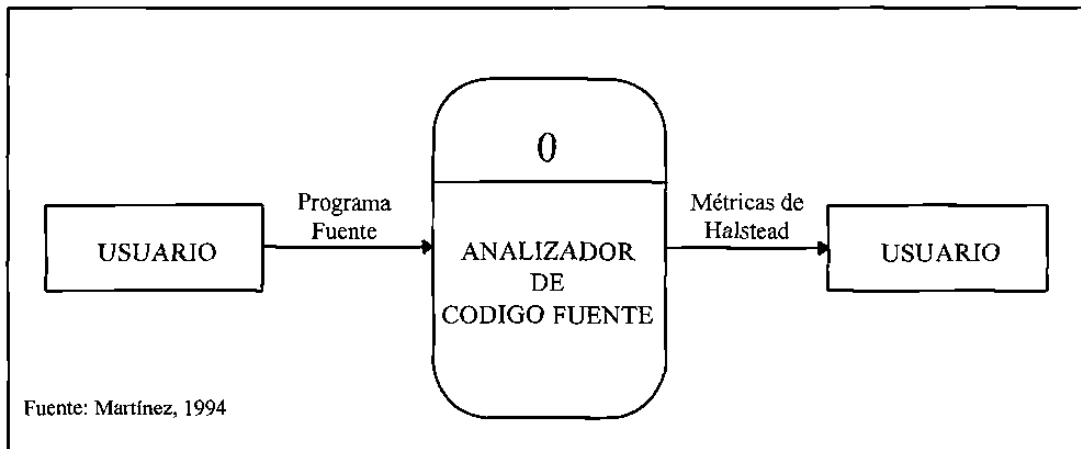


Figura 11. Diagrama de Flujo de Datos del Analizador de Código (Nivel 0)

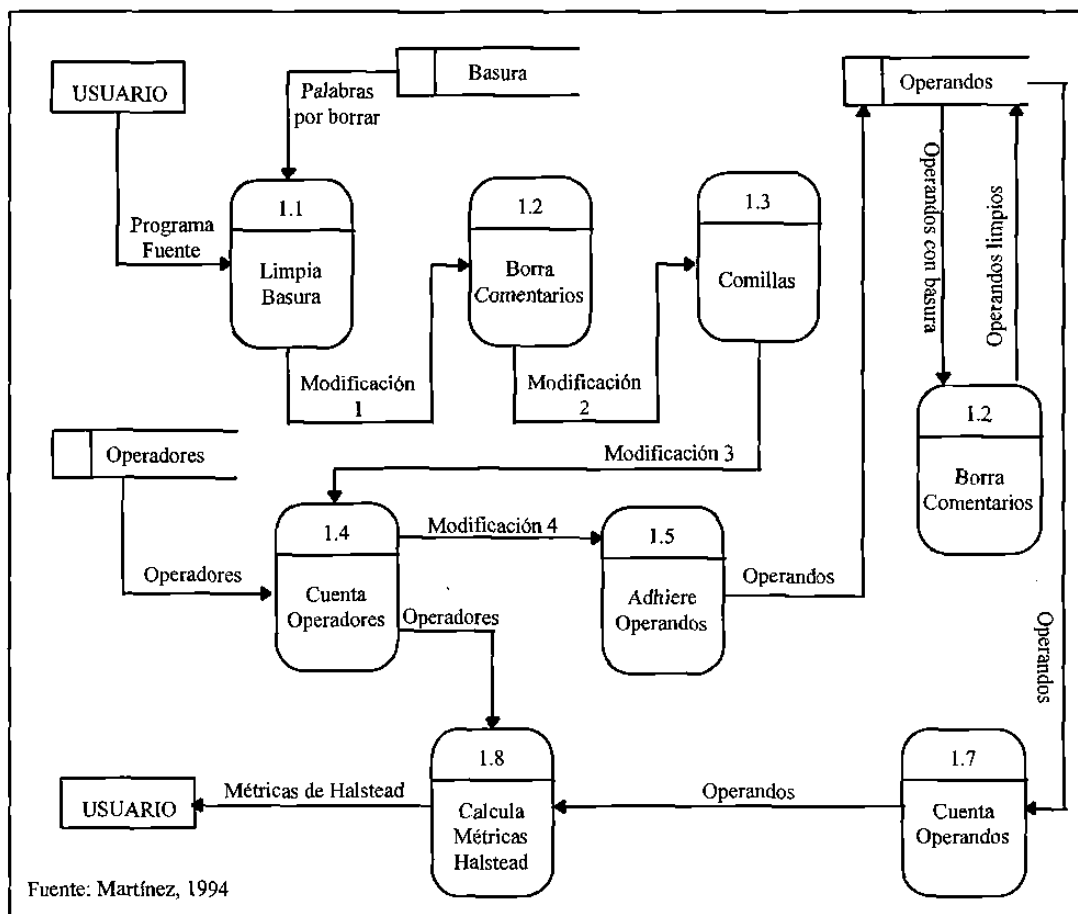


Figura 12. Diagrama de Flujo de Datos del Analizador de Código (Nivel 1)

Por último, para las personas que deseen utilizar el analizador de código para aplicarlo en otros lenguajes de programación, el disco que se anexa a la presente tesis contiene todos los programas en código fuente del analizador, de tal forma que puede ser modificado para emplearse en otros lenguajes de programación que no sean los LMBD PROGRESS6 y ORACLE7. Las modificaciones básicas que deben realizarse al analizador de código para emplearse en otros lenguajes son:

1. Construir un archivo de operadores del lenguaje a analizar.
2. Construir un archivo de basura, el cual contiene los operadores que no intervienen en el conteo al obtener las Métricas de Halstead.
3. Modificar el código fuente para eliminar los comentarios cuando éstos se especifiquen en forma diferente a como se especifican en los LMBD PROGRESS6 y ORACLE7.

RESUMEN AUTOBIOGRAFICO

Luis Gerardo Navarro Guerra

Candidato para el Grado de

Maestro en Ciencias de la Administración con Especialidad en Sistemas

Tesis: METRICAS DE HALSTEAD EN LENGUAJES MANIPULADORES DE BASES DE DATOS

Campo de Estudio: Métricas de Software

Biografía:

Nacido en Monterrey, Nuevo León, el 1 de Noviembre de 1973; hijo de Julio Navarro Mercado y María Ana Guerra Pérez.

Educación:

Egresado de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León; grado obtenido de Ingeniero Administrador de Sistemas en 1995.

