UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POSTGRADO

METRICAS DE HALSTEAD APLICADAS A LENGUAJES DE PROGRAMACION ORIENTADOS A OBJETOS

TESIS

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

POR

ING. XAVIER ESPINOSA DE LOS MONTEROS ANZALDUA

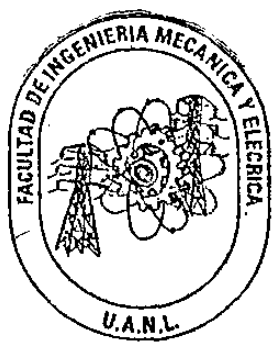
SAN NICOLAS DE LOS GARZA, N. L. ENERO DE 1997

METRICAS DE HALSTEAD APLICADAS A LENGUAJES DE PROGRAMACION ORIENTADOS A OBJETOS

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POSTGRADO



METRICAS DE HALSTEAD APLICADAS A LENGUAJES DE PROGRAMACION ORIENTADOS A OBJETOS

T E S I S

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

P O R

ING. XAVIER ESPINOSA DE LOS MONTEROS ANZALDUA

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

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA
DIVISIÓN DE ESTUDIOS DE POSTGRADO

Los miembros del comité de tesis recomendamos que la tesis MÉTRICAS DE HALSTEAD APLICADAS A LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS realizada por el Ing. Xavier Espinosa de los Monteros Anzaldúa 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:

Enrique Espinosa de los Monteros Martínez y María Elma Anzaldúa de Espinosa de los Monteros, gracias por su amor, apoyo y comprensión durante toda mi vida; por guiarme con su ejemplo; nunca olviden que los amo.

A mi tío:

Dr. Sigifredo Anzaldúa Hernández, por todo el apoyo y comprensión brindados para poder llegar hasta aquí.

A mi hermano:

Enrique, por su ayuda en los momentos en que lo necesité.

A mis compañeros y amigos.

Por su amistad y por toda la ayuda que desinteresadamente me han brindado en todo momento.

AGRADECIMIENTOS

Deseo expresar mi más sincero agradecimiento al Dr. José Luis Martínez Flores, asesor de esta tesis, y a los coasesores, Dra. Ada Margarita Álvarez Socarrás y Dr. Oscar Flores Rosales por su valiosa ayuda, consejos, recomendaciones y sus acertadas guías para el desarrollo del presente trabajo.

Deseo agradecer de manera especial a todos mis compañeros y amigos del Doctorado de Ingeniería de Sistemas, por su invaluable amistad, apoyo y compañía que me brindaron durante estos años.

Agradezco a la Facultad de Ingeniería Mecánica y Eléctrica y a la Universidad Autónoma de Nuevo León toda la ayuda proporcionada.

Por último, agradezco al Consejo Nacional de Ciencia y Tecnología por su apoyo para realizar el postgrado.

Resumen

Xavier Espinosa de los Monteros Anzaldúa Fecha de Graduación: Enero, 1997

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Título del Estudio: MÉTRICAS DE HALSTEAD APLICADAS A LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

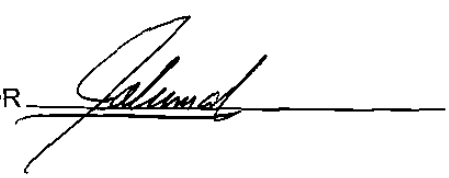
Número de Páginas: 91

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

Área de Estudio: Métricas de Software

Propósito y Método del Estudio: El propósito principal de esta investigación es determinar si las métricas de software que propuso Halstead en 1977 (principalmente el estimador de la longitud de un programa y el estimador del nivel del lenguaje) son válidas para los lenguajes de programación orientados a objetos. Estas métricas han sido probadas en los lenguajes máquina, en los lenguajes ensambladores, en los lenguajes de tercera generación y en los lenguajes de cuarta generación; con los cuales se han obtenido buenos resultados. En esta investigación se utilizó un analizador de código desarrollado en una investigación anterior con algunas modificaciones para analizar el lenguaje de programación orientado a objetos C++ versión 3.1. La muestra que se utilizó fueron los programas que vienen de ejemplo en el paquete.

Contribuciones y Conclusiones: Los resultados de la investigación fueron (1) el estimador de la longitud propuesto por Halstead sí es un buen estimador para el lenguaje de programación orientado a objetos C++, versión 3.1 y (2) el nivel del lenguaje para el C++ fue de 1.84348, el cual está entre los valores del nivel del lenguaje para los lenguajes de tercera generación de propósito general y los lenguajes de cuarta generación. Este resultado se puede agregar a la tabla de clasificación que realizó Halstead. Con estos resultados podemos obtener una metodolgía para la evaluación de software desarrollado en C++, con ésta podemos evaluar el desempeño de programadores que desarrollen software en C++, también se puede evaluar el desempeño de alumnos de escuelas de programación con el objetivo de obtener una medida de comparación para las diferentes escuelas.

FIRMA DEL ASESOR 

NOMENCLATURA

3GL	Lenguajes de Tercera Generación.
4GL	Lenguajes de Cuarta Generación
LPOO	Lenguajes de Programación Orientados a Objetos.
PPE	Programación Procedural Estructurada.
POO	Programación Orientada a Objetos.
DOO	Diseño Orientado a Objetos.
OO	Orientado a Objetos.
AOO	Análisis Orientado a Objetos.

TABLA DE CONTENIDO

Capítulo **Página**

Capítulo **Página**

Capítulo **Página**

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1 Establecimiento del Problema

Es muy común que los analistas y diseñadores de sistemas realicen sus propias estimaciones de software en base a su experiencia. Si el analista tiene mucha experiencia, los resultados son satisfactorios, pero deja mucho que desear la formalidad con que se hacen las estimaciones de software.

A algunos analistas y diseñadores de sistemas sólo les interesa que el software funcione y no les importa cómo se desarrolló. Esto puede causar muchos problemas cuando el software requiera mantenimiento, demostrando de este modo la baja calidad del mismo.

Con lo anterior, sería conveniente encontrar alguna forma de medir la calidad del software que se desarrolla, así como estimar su tamaño y tiempo de desarrollo.

Actualmente el paradigma de la orientación a objetos está llegando a ser muy popular, es fácil darse cuenta al leer bibliografía relacionada con el tema. Uno de los lenguajes de programación orientados a objetos que se menciona muy frecuentemente es el lenguaje C++; por lo cual es necesario tener alguna medida que

nos sea de utilidad para comparar el C++ con otros lenguajes. Para obtener esta medida de comparación se pueden aplicar las métricas de Halstead.

1.1.1 Métricas de Software

Antes de mencionar las métricas de software primero nos debemos relacionar con la definición de la ingeniería del software.

La ingeniería del software 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 y mejores técnicas para la garantía de la calidad del software, y una filosofía predominante para la coordinación, control y administración [PRES93].

Otro concepto que debemos tener presente en este tema es la calidad del software, la cual está definida como "la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente", [PRES93].

La medición es fundamental para cualquier disciplina de la ingeniería y en la ingeniería del software no es una excepción, existiendo varias razones por las cuales medir el software [PRES93]:

1) Para indicar la calidad del producto.

2) Para evaluar la productividad de la gente que desarrolla el producto.

3) Para evaluar los beneficios derivados del uso de nuevos métodos y herramientas de la ingeniería del software.

4) Para establecer una línea base para la estimación.

5) Para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

Los factores que afectan la calidad del software se clasifican en dos categorías: a) factores que pueden ser medidos directamente y b) factores que sólo pueden ser medidos indirectamente. Las métricas cualitativas son aquellas que miden el software en forma indirecta o subjetiva. Las métricas cuantitativas son aquellas que miden el software en forma directa u objetiva.

Existen muchos factores que afectan el desarrollo de un programa, éstos incluyen: el tipo de programa a desarrollar, el tamaño del programa, el lenguaje de implementación, la experiencia de los programadores, las técnicas de programación y el ambiente computacional.

Dentro de las métricas cuantitativas se encuentra la Ciencia del Software [HALS77]. La Ciencia del Software es un modelo del proceso de programación que se basa en un número manipulable de los principales factores que afectan la programación. Esto ofrece una guía hacia estimadores que pueden ser útiles a los administradores de proyectos de software.

1.1.2 Orientación a Objetos

La orientación a objetos está ganando popularidad en el mundo académico, en los negocios y en la industria. Esta tecnología se inició en Noruega en los 60´s, donde los científicos desarrollaron un lenguaje llamado Simula. Un equipo investigador de la Xerox continuó este esfuerzo y desarrolló un lenguaje orientado a objetos llamado Smalltalk, el cual es ahora muy usado, particularmente en las instituciones académicas. La tecnología de objetos consiste en un diseño, desarrollo y bases de

datos orientadas a objetos. La programación orientada a objetos y la representación del conocimiento orientado a objetos son también parte de esta tecnología [FREN92].

El paradigma de la orientación a objetos enfoca problemas desde un nivel de abstracción diferente al de la programación convencional. Existen muchos desarrollos nuevos en esta tecnología que está emergiendo rápidamente, y está siendo ampliamente utilizada. Muchos desarrolladores están utilizando C++, una versión de C con capacidades de orientación a objetos. La tecnología de objetos puede ser en los 90´s lo que las técnicas estructuradas fueron en los 80´s [FREN92].

Es de vital importancia para cualquier empresa o institución educativa conocer y entender el funcionamiento de las nuevas herramientas y metodologías; la tecnología orientada a objetos es un nuevo enfoque en el desarrollo de software, el cual según algunos autores entre ellos Yourdon, llegará a ser de gran utilidad en muchas empresas, las cuales deberán ante todo tener un cambio cultural más que tecnológico [YOUR94].

1.2 Objetivo de la Investigación

La tercera generación de lenguajes está dividida en tres categorías, las cuales son: (1) lenguajes de alto nivel de propósito general, (2) lenguajes de alto nivel orientados a objetos y (3) lenguajes especializados [PRES93].

Los resultados de la Ciencia del Software se han aplicado a diferentes generaciones de lenguajes, tales como el lenguaje máquina, los lenguajes ensambladores, los lenguajes de tercera generación (3GL´s) de propósito general y un estudio más reciente en lenguajes de cuarta generación (4GL´s). Por lo tanto, nos hacemos la siguiente pregunta: ¿Los resultados de la Ciencia del Software tienen sentido en los lenguajes de programación orientados a objetos (LPOO´s)?

Para tratar de contestar a la pregunta anterior se analiza uno de los estimadores que propone Halstead y se clasifica un lenguaje de programación orientado a objetos dentro de la tabla de clasificaciones de Halstead.

Halstead [HALS77] propone un estimador de la longitud de un programa ($\hat{N}$), del cual existe evidencia empírica que demuestra que es un buen estimador para lenguajes de tercera [SHEN83] y cuarta generación [MART94], pero ¿Es $\hat{N}$ un buen estimador de la longitud de un programa para los LPOO´s?

Halstead hace una clasificación de diferentes niveles de lenguaje (Inglés Prosaico, PL/1, Algol 58, Fortran, Pilot, Ensamblador). ¿Tiene sentido clasificar los LPOO´s, como lo hizo Halstead? Esto es, ¿Es el nivel del lenguaje de los LPOO´s mayor que el nivel del lenguaje de los 3GL´s y menor que el nivel del lenguaje de los 4GL´s?

La presente tesis intenta dar respuesta a las preguntas anteriores mediante el logro de los siguientes objetivos:

1) Determinar si el estimador de la longitud de un programa, es un buen estimador para lenguajes de programación orientados a objetos.

2) Determinar si el nivel del lenguaje de los LPOO´s es mayor que el nivel del lenguaje de los 3GL´s de propósito general y menor que el nivel del lenguaje para los 4GL´s.

1.3 Resumen

En este capítulo se presentó el establecmiento del problema, se habló un poco de las métricas de software y del paradigma de la orientación a objetos. También se presentó el objetivo de la investigación.

CAPÍTULO 2

ANTECEDENTES

2.1 Introducción

El objetivo principal de la tesis es probar si las métricas de Halstead aún siguen siendo efectivas para la medición de lenguajes de programación orientados a objetos. En 2.2 se habla del concepto del software, su evolución, sus características, su desarrollo, sus problemas y las causas que los originan. En 2.3 se habla de la ingeniería del software. En 2.4 se habla de la medición del software y en 2.5 se habla de la calidad del software, dentro de la cual están las métricas de Halstead.

2.2 Software

En las primeras tres décadas de la Informática, el principal desafío era el desarrollo de hardware de las computadoras, de forma que se redujera el costo de procesamiento y almacenamiento de los datos. Hoy, el principal desafío es mejorar la calidad (y reducir el costo) de las soluciones basadas en computadoras, es decir, soluciones que se implementan con el software [PRES93].

2.2.1 Concepto del Software

Una descripción del software puede tener la siguiente forma: (1) instrucciones (programas de computadora) que cuando se ejecutan proporcionan la función y el comportamiento deseado, (2) estructuras de datos que facilitan a los programas manipular adecuadamente la información, y (3) documentos que describen la operación y el uso de los programas [PRES93].

Software: Son los programas, o conjuntos de instrucciones, que dirigen la operación del hardware de una computadora [ATHE88].

Software: Es el término general que describe a los programas de instrucciones, lenguajes, o rutinas o procedimientos que hacen posible a una persona usar una computadora [JAME87].

Independientemente de la descripción del software dada por diferentes autores, el software es el conjunto de instrucciones o programas que le indican a una computadora las operaciones que va a realizar.

2.2.2 Evolución del Software

La figura 1 describe la evolución del software [PRES93]. Durante los primeros años de desarrollo de las computadoras, el hardware sufrió continuos cambios, mientras que el software se contemplaba simplemente como un añadido. La programación de las computadoras era un arte para el que existían pocos métodos sistemáticos. El desarrollo de software se realizaba sin ninguna planificación. Durante este período se utilizaba en la mayoría de los sistemas una orientación por lotes.

Durante los primeros años, el software se diseñaba a la medida para cada aplicación. La mayoría del software se desarrollaba y era utilizado por la misma

persona u organización. La misma persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba. Debido a este entorno personalizado del software, el diseño era algo implícito, y la documentación normalmente no existía. A lo largo de los primeros años se aprendió poco sobre la ingeniería de las computadoras.

En la segunda era de la evolución del software, la multiprogramación y los sistemas multiusuario introdujeron nuevos conceptos de interacción hombre-máquina. Las técnicas interactivas abrieron un nuevo mundo de aplicaciones y nuevos niveles de sofisticación del hardware y del software. En esta era de la evolución del software surgieron los sistemas de tiempo real y los sistemas de administración de bases de datos, ésta también se caracterizó por el establecimiento del software como producto y la llegada de las casas de software. El software ya se desarrollaba para tener una amplia distribución en un mercado multidisciplinario. Aquí surgió el concepto de *mantenimiento del software* que se refiere a la necesidad de modificar las sentencias fuente cuando se detectaban fallas, o cuando los requisitos de los usuarios cambiaban. La naturaleza personalizada de muchos programas los hacía imposibles de mantener. Comenzó una crisis del software.

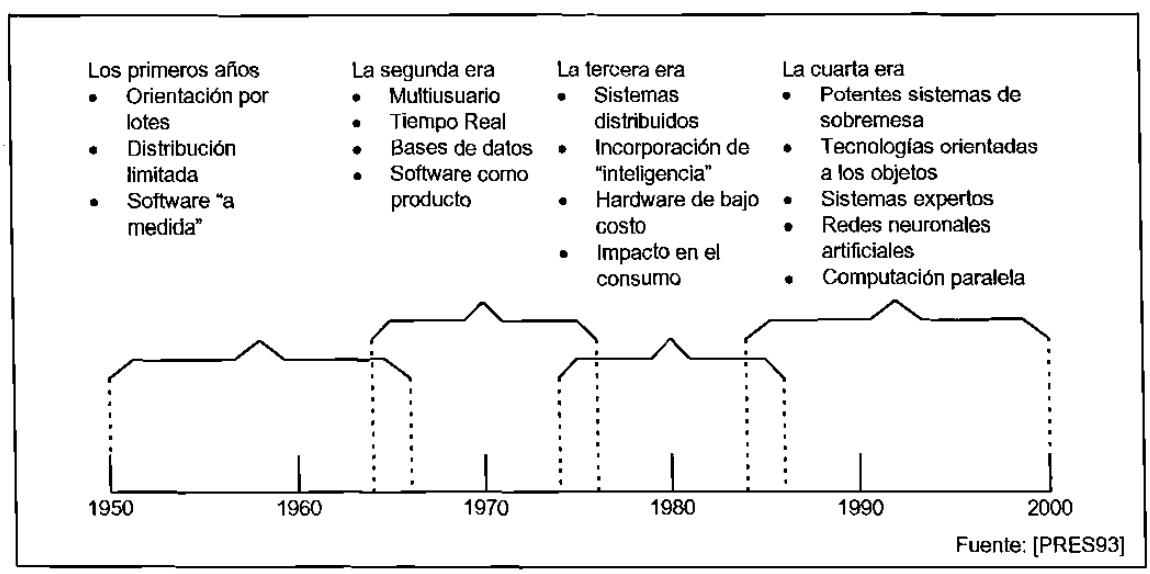


Figura 1 Evolución del Software.

La tercera era se caracteriza también por la llegada y el amplio uso de los microprocesadores y las computadoras personales. Las computadoras personales han sido el catalizador del crecimiento de las compañías de software. El hardware de las computadoras personales se ha convertido en un producto estándar, mientras que el software que se suministre con ese hardware, es lo que marca la diferencia.

La cuarta era de la evolución del software de computadora está comenzando. Las tecnologías orientadas a los objetos están desplazando a los enfoques de desarrollo de software convencionales en muchas áreas de aplicación. Las técnicas de cuarta generación para el desarrollo de software ya están cambiando la forma en que algunos segmentos de la comunidad informática construyen los programas de computadora. Los sistemas expertos y el software de inteligencia artificial se han trasladado del laboratorio a las aplicaciones prácticas.

2.2.3 Características del Software

Para comprender lo que es el software, es importante examinar las características del mismo que lo diferencian de otras cosas que los hombres pueden construir.

Características del software [PRES93]:

1) *El software se desarrolla, no se fabrica en un sentido clásico.*

2) *El software no se estropea.* El software no es susceptible a los males del entorno que hacen que el hardware se estropee (ver figuras 2 y 3).

3) *La mayoría del software se construye a la medida, en vez de ensamblar componentes existentes.*

Esta última característica ha ido desapareciendo ya que la reutilización del software es uno de los principales contribuidores técnicos para la productividad y calidad del software [YOUR94]. También las técnicas orientadas a objetos ofrecen una alternativa para escribir los mismos programas una y otra vez [WINB93].

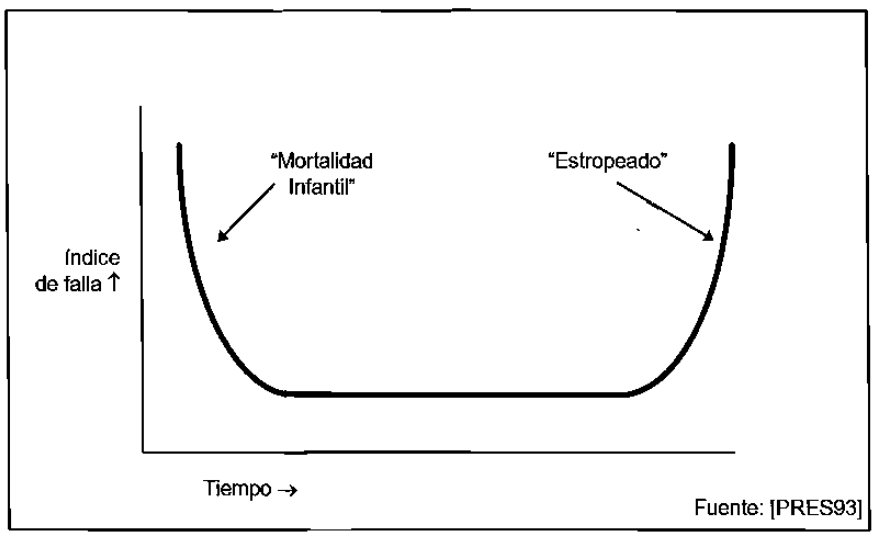


Figura 2 Curva de Fallas del Hardware.

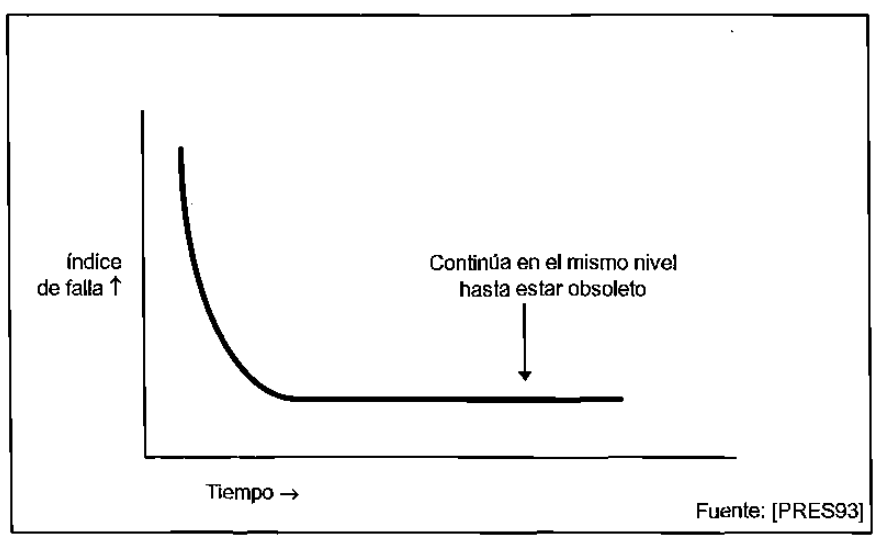


Figura 3 Curva de Fallas del Software.

2.2.4 Desarrollo del Software

La reutilización es una característica importante para un software de alta calidad. Un software reutilizable encapsula tanto datos como procesos en un paquete único (clase u objeto), permitiendo crear nuevas aplicaciones a partir de software reutilizable.

El software se desarrolla mediante un lenguaje de programación que tiene un vocabulario limitado, una gramática definida explícitamente y reglas bien formadas de sintaxis y semántica. Las clases de lenguajes que se utilizan son los lenguajes máquina, los lenguajes de alto nivel y los lenguajes no procedimentales.

Los lenguajes máquina son una representación simbólica del conjunto de instrucciones del CPU. Los lenguajes de alto nivel tales como COBOL, FORTRAN, Pascal, C, Ada, C++, Object Pascal, etc. permiten al programador y al programa interactuar normalmente sin importar el equipo de cómputo que se utilice. Estos lenguajes se dividen en tres categorías que son: (1) de propósito general, (2) orientados a objetos y (3) lenguajes especializados. De los lenguajes de alto nivel anteriormente mencionados, C++ y Object Pascal pertenecen a la categoría de orientados a objetos.

El código máquina, los lenguajes ensambladores y los lenguajes de programación de alto nivel son considerados como las tres primeras generaciones de los lenguajes de computadora. Con estos lenguajes, el programador debe preocuparse tanto de la especificación de la estructura de la información como de la de control del propio programa. Por ello, los lenguajes de las tres primeras generaciones se denominan *lenguajes procedimentales.*

Los lenguajes *no procedimentales* o de cuarta generación aparecieron en la década pasada. En los lenguajes de programación de alto nivel se requiere que se

especifiquen los detalles procedimentales, y en los no procedimentales únicamente se especifica el resultado deseado, en vez de especificar la acción requerida para conseguir ese resultado [PRES93].

2.2.5 Problemas con el Desarrollo del Software

Los problemas del desarrollo de software se centran en los siguientes aspectos [PRES93]:

1) La planificación y estimación de costos son frecuentemente muy imprecisas.

2) La productividad de la comunidad de software no satisface la demanda de sus servicios.

3) La calidad del software a veces es inaceptable.

2.2.6 Causas de los Problemas del Desarrollo del Software

Los problemas asociados con el desarrollo de software se han producido por la propia naturaleza del software y por los errores de las personas encargadas del desarrollo del mismo.

El software es un elemento lógico en vez de físico, por tanto el éxito se mide por la calidad de una única entidad en vez de muchas entidades fabricadas. El software no se rompe, si se encuentran fallas, lo más probable es que se introdujeran inadvertidamente durante el desarrollo y no se detectaran durante la prueba.

Los programadores han tenido muy poco entrenamiento formal en las nuevas técnicas de desarrollo de software. En algunas organizaciones, cada individuo enfoca su tarea de escribir programas con la experiencia obtenida en trabajos anteriores. Algunas personas desarrollan un método ordenado y eficiente de desarrollo del

software mediante prueba y error, pero otros desarrollan malos hábitos que dan como resultado una pobre calidad y mantenibilidad del software [PRES93].

2.3 La Ingeniería del Software

No existe el mejor enfoque para solucionar los problemas del software. Sin embargo, mediante la combinación de métodos completos para todas las fases del 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 gestión, podemos conformar una disciplina para el desarrollo del software, *ingeniería del software* [PRES93].

2.3.1 Definición

Una definición de la ingeniería del software propuesta por Fritz Bauer, citado por Pressman [PRES93], es:

"El establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico que sea fiable y funcione de manera eficiente sobre máquinas reales".

La ingeniería del software abarca un conjunto de tres elementos clave: *métodos, herramientas* y *procedimientos,* que facilitan al administrador controlar el proceso del desarrollo del software y suministrar a los que practiquen dicha ingeniería las bases para construir software de alta calidad de una forma productiva.

Los *métodos* de la ingeniería del software indican "cómo" construir técnicamente el software. Los métodos incluyen tareas de: planificación 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.

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 de software, llamado *ingeniería del software asistida por computadora* (CASE).

Los *procedimientos* de la ingeniería del software definen la secuencia en la que se aplican los métodos, las entregas (documentos, informes, formas) que se requieren, los controles que ayudan a asegurar la calidad y coordinar las cambios, y las directrices que ayudan a los administradores del software a evaluar el progreso.

2.4 Medición de Software

La medición está definida como el proceso mediante el cual se asignan números o símbolos a los atributos o entidades del mundo real de tal forma que lo describan de acuerdo con reglas claramente definidas [FENT94].

La medición es fundamental para cualquier disciplina de ingeniería y la ingeniería del software no es la excepción. La medición es muy común en el mundo de la ingeniería. Medimos potencias de consumo, pesos, dimensiones físicas, temperaturas, voltajes, señales de ruido, etc. Desgraciadamente, la medición se aleja de lo común en el mundo de la ingeniería del software. Encontramos dificultades en ponernos de acuerdo sobre qué medir y cómo evaluar las medidas [PRES93]. Fenton [FENT94] menciona que el proceso de medición del software tiene dos grandes usos: para evaluar y para predecir.

Por varios años la medición de la calidad y productividad del software fue difícil y muy pocas compañías la realizaban. Pero existen métricas estables y exactas a partir de 1979, y ahora cada compañía puede obtener los beneficios de la aplicación de la medición del software [JONE91].

Uno de los problemas con la medición del software no es una deficiencia en la medición, sino que es una resistencia cultural por parte de los administradores y personal técnico del software. La resistencia se debe a que la naturaleza humana cree que las mediciones pueden ser usadas en su contra. Este sentimiento crea una barrera para la aplicación de la medición. El reto es romper con esa barrera y demostrar que la aplicación de las mediciones no son dañinas, sino necesarias para el éxito corporativo [JONE91].

2.4.1 El Problema del Software

Mejorar la calidad y desempeño del producto de software y la productividad del equipo de desarrollo es la prioridad principal para la mayoría de las organizaciones. Como las computadoras son cada vez más poderosas, los usuarios demandan software más poderoso y sofisticado.

El problema del software es grande. Muy pocos sistemas grandes han sido terminados dentro del presupuesto, del tiempo y que hayan cubierto todos los requerimientos del usuario. Además, el promedio de los proyectos de sistemas grandes terminan un año después y con el doble del costo estimado inicialmente [MÖLL93].

Un problema muy importante del software es estimar su costo, pero para poder realizarlo se necesita tener un estimado exacto del tamaño del software que va a ser desarrollado. Para esto se han desarrollado algunos métodos de estimación del

tamaño del software para varios lenguajes de programación [LOKA96], [VERN92], [WRIG91].

Uno de los aspectos más frustrantes del problema de desarrollo de software es el gran número de proyectos que son entregados a los clientes después de tiempo. Esto resulta en pérdidas de oportunidades y clientes insatisfechos [MÖLL93].

Los problemas con la calidad y desempeño del software pueden afectar las relaciones de negocios con los clientes. Las fallas no descubiertas son frecuentemente encontradas por los clientes después de la entrega del producto. La probabilidad de que esto ocurra puede ser minimizada por la implementación de técnicas de administración de la calidad del software dentro del proceso de desarrollo de software [MÖLL93].

Muchas compañías están descubriendo que el problema del software es tanto un problema de administración como un problema técnico. La ingeniería del software es relativamente una nueva disciplina técnica. Muchas de las técnicas de administración y de aseguramiento de la calidad utilizadas en otras disciplinas de ingeniería y producción también son aplicables al desarrollo del software [MÖLL93].

Los métodos estructurados, como algunas veces se refieren a la ingeniería del software, responden a los problemas de una pobre calidad del software, dificultad de mantenimiento, y baja productividad del programador [JONH90].

2.4.2 Razones para Medir el Software

Existen varias razones para medir el software y son las siguientes [PRES93]:

1) Para indicar la calidad del producto.

2) Para evaluar la productividad de la gente que desarrolla el producto.

3) Para evaluar los beneficios (en términos de productividad y de calidad) derivados del uso de nuevos métodos y herramientas de ingeniería del software.

4) Para establecer una línea de base para la estimación.

5) Para ayudar a justificar el uso de nuevas herramientas o de formación adicional.

2.4.3 Clasificación de las Mediciones del Software

Las mediciones pueden clasificarse en dos categorías: *mediciones directas* y *mediciones indirectas*. Entre las mediciones directas se encuentran el costo y el esfuerzo aplicado, las líneas de código producidas, la velocidad de ejecución, etc. Entre las medidas indirectas se encuentran la funcionalidad, la calidad, complejidad, eficiencia, etc. [PRES93].

Podemos clasificar las métricas del software como se muestra en la figura 4. Las *métricas de productividad* se centran en el rendimiento del proceso de la ingeniería del software, las *métricas de calidad* proporcionan una indicación de cómo se ajusta el software a los requisitos implícitos y explícitos del cliente y las *métricas técnicas* se centran en las características del software más que en el proceso a través del cual ha sido desarrollado.

Las *métricas orientadas al tamaño* se utilizan para obtener medidas directas del resultado y de la calidad de la ingeniería del software. Las *métricas orientadas a la función* proporcionan medidas indirectas y las *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.

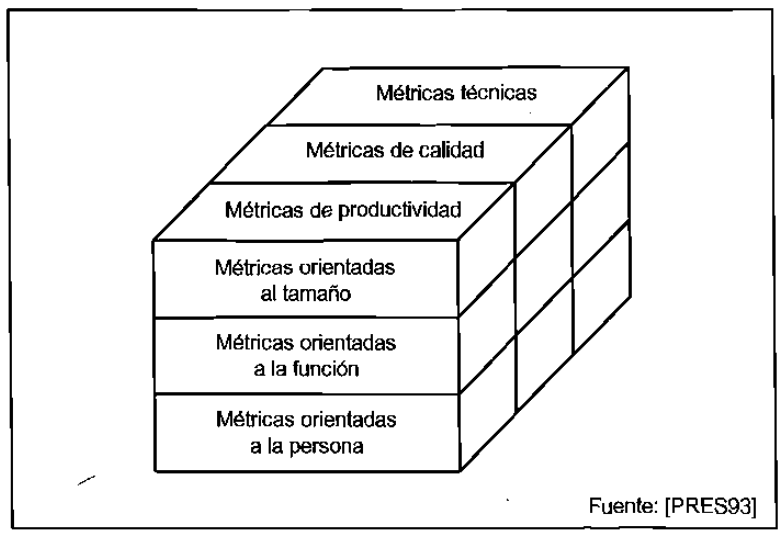


Figura 4 Clasificación de Métricas.

2.4.4 Beneficios

La implementación de un programa de introducción de métricas resultará en muchos beneficios para las organizaciones. Estos beneficios incluirán costos bajos de desarrollo como resultado de una alta productividad, altas ventas debido a ciclos de desarrollo más cortos los cuales son resultado de productos de más alta calidad. El uso de métricas mejorará la capacidad de planear proyectos nuevos. Cuando existen datos históricos de proyectos, se pueden hacer comparaciones entre los proyectos nuevos y proyectos anteriores similares. Esto mejorará la capacidad de estimar costos y tiempos de desarrollo para los proyectos nuevos [MÖLL93].

El programa de introducción de métricas resultará en un incremento en la confianza del empleado, por la demostración de que la compañía tiene buenos conocimientos de las fortalezas y debilidades de sus productos y procesos de desarrollo, y que está tomando acciones positivas para corregir sus debilidades.

Se debe notar que el programa por sí sólo no producirá todos los beneficios. El programa de métricas es sólo una ayuda para mejorar el proceso de desarrollo de

software. El desarrollo productivo de productos de alta calidad dependerá de qué tan bien sean implementados y administrados los procesos.

Algunos de estos beneficios se observan en un estudio realizado por Bowman [BOWM90], entre los cuales figuran: software de mejor calidad, menor tiempo de desarrollo, menor complejidad, etc.

2.5 Calidad del Software

Una definición de calidad comúnmente utilizada es la siguiente:

"Es la totalidad de las características de un producto o servicio que tienen la capacidad de satisfacer las necesidades implicadas" [MÖLL93].

Pressman [PRES93] define la calidad del software como:

Concordancia con 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.

La definición anterior hace énfasis en tres puntos importantes:

1) Los requisitos del software son la base de las medidas de 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 del software. Si no se siguen esos criterios, casi siempre habrá falta de calidad.

3) Existe un conjunto de requisitos implícitos que a menudo no se mencionan (tal como el deseo de un buen mantenimiento). Si el software cumple con

sus requisitos explícitos pero falla al alcanzar los requisitos implícitos, la calidad del software queda en entredicho.

El desarrollador está interesado en aprender cómo desarrollar un producto que exhiba una buena calidad. Para el desarrollador de software es necesario identificar los aspectos de calidad que pueden ser reconocidos por su presencia o ausencia. Las métricas son una herramienta que ayuda a cuantificar aspectos de calidad de tal forma que se puedan medir las acciones necesarias para mejorarla [MÖLL93].

Otro aspecto de la calidad que es importante entender es que ésta es una característica central que tiene efecto sobre otras características del producto de software y el proceso de desarrollo (figura 5). Mejorando la calidad se tendrá una mejora secundaria en la funcionalidad del producto y en el esfuerzo y tiempo de implementación del mismo [MÖLL93].

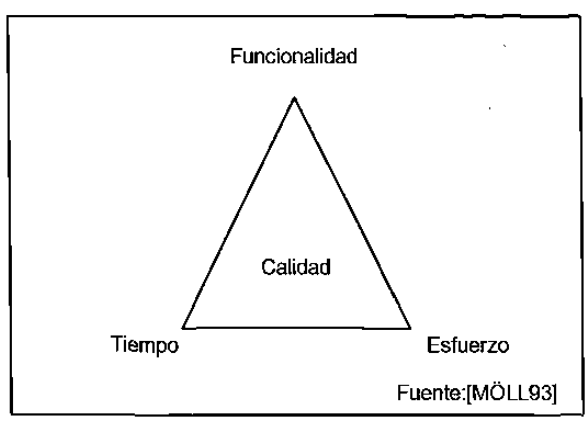


Figura 5 Impactos de la Calidad.

Existe un conjunto de 5 pasos para controlar la calidad del software [JONE91]:

1) Establecer un programa de métricas de calidad del software.

2) Establecer metas tangibles de desempeño del software ejecutivo.

3) Establecer una evaluación de la calidad del software significativo.

4) Desarrollar una cultura corporativa de vanguardia.

5) Determinar las fortalezas y debilidades del software.

2.5.1 Factores que Determinan la Calidad del Software

Los factores que afectan la calidad del software se pueden clasificar en dos grandes grupos:

1) Factores que pueden ser medidos directamente (por ejemplo, errores/líneas de código/unidad de tiempo).

2) Factores que sólo pueden ser medidos indirectamente (por ejemplo, facilidad de uso o mantenimiento).

McCall citado por Martínez [MART94] ha propuesto una clasificación de los factores que afectan la calidad del software. Estos factores de la calidad del software, que aparecen en la figura 6, se centran en tres aspectos importantes de un producto de software: sus características operativas, su capacidad de soportar los cambios y su adaptabilidad a nuevos entornos.

Es difícil, y en algunos casos imposible, desarrollar medidas directas de los anteriores factores de calidad. Por tanto, se define un conjunto de métricas que se utilizan para medir de forma indirecta los factores de calidad del software. Existen dos tipos de métricas [PRES93]:

1) *Métricas cualitativas.* Estas métricas sólo pueden ser medidas en forma subjetiva.

2) *Métricas cuantitativas.* Estas métricas si pueden medirse en forma objetiva.

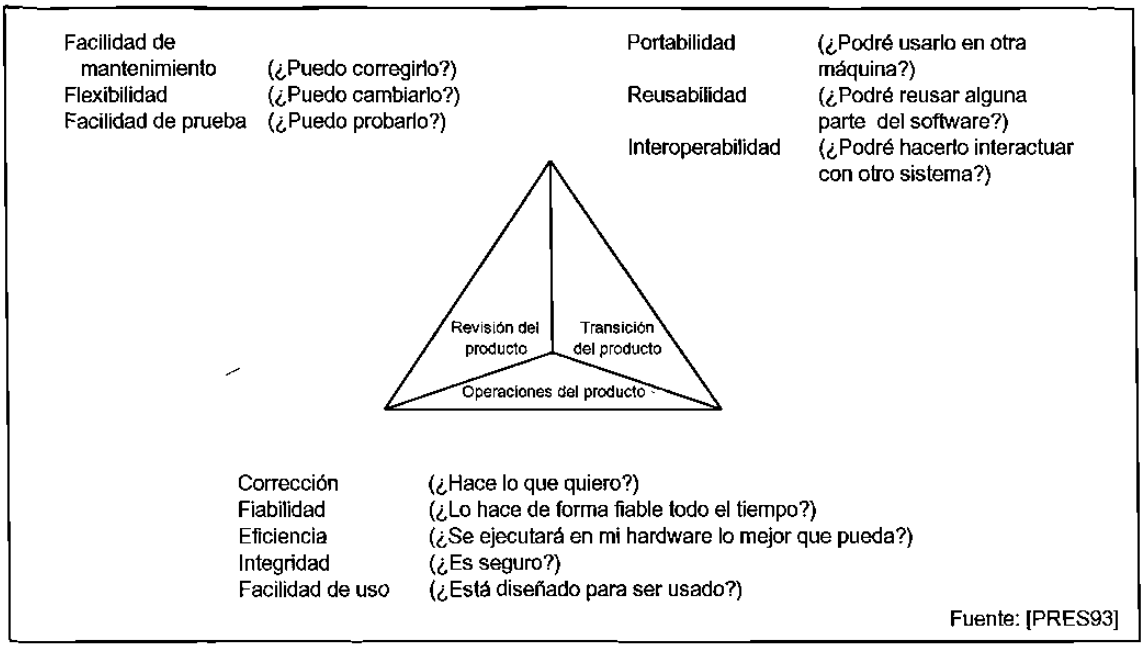


Figura 6 Factores de la Calidad del Software de McCall.

2.5.2 Métricas Cualitativas de la Calidad del Software

McCall citado por Martínez [MART94] ha definido algunas métricas que sólo pueden ser medidas en forma subjetiva, entre las cuales figuran las siguientes:

Facilidad de auditoría. La facilidad con que se puede comprobar la conformidad con 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 del proyecto de desarrollo del software.

Estandarización en los datos. El uso de estructuras de datos y de tipos estándar a lo largo de todo el programa.

Tolerancia de 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 ampliar 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 que 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 controlen o protejan los programas o los datos.

Autodocumentación. El grado en que el código fuente proporciona documentación significativa.

Simplicidad. El grado en que un programa puede ser entendido sin dificultad.

Independencia del sistema de software. El grado en que el programa es independiente de características no estándar del lenguaje de programación, de las características 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 de la Calidad del Software

En esta sección se verán algunas métricas que se pueden aplicar para garantizar cuantitativamente la calidad del software.

2.5.3.1 La Ciencia del Software

La teoría de Halstead sobre la ciencia del software [HALS77] asigna leyes cuantitativas al desarrollo de software de computadora. La teoría de Halstead se deriva de una suposición fundamental: "El cerebro humano sigue un conjunto de reglas más rígido (al desarrollar algoritmos) de lo que nunca ha tenido consciencia...". La ciencia del software usa 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.

Las métricas de Halstead se basan en partículas elementales de los programas, tales como operadores y operandos; estas métricas se explicarán con mayor detalle en el capítulo 3. Además existe evidencia empírica de que son válidas para los lenguajes de tercera [SHEN83] y de cuarta generación [MART94].

Tambíen existen otras métricas que se pueden aplicar para determinar la calidad del producto de software, tales como la complejidad ciclomática de McCabe [MCCA76] y los indicadores de calidad del software desarrollados por el US Air Force Command [PRES93].

2.6 Resumen

En este capítulo se presentaron los conceptos relacionados con el software, su evolución, su calidad y las métricas para medirla. Se observó que las métricas de Halstead, las cuales se verán con mayor detalle en el siguiente capítulo, se encuentran dentro de la clasificación de las métricas cuantitativas de la medición del software.

CAPÍTULO 3

MÉTRICAS DE HALSTEAD

3.1 Introducción

En esta sección se presentan las propiedades básicas de las Métricas de Halstead. En 3.2 se habla acerca de la Ciencia del Software. En 3.3 se trata la longitud del programa y su estimador. En 3.4 se trata el volumen del programa y su estimador. En 3.5 se trata el nivel y dificultad de un programa y su estimador. En 3.6 se trata el contenido de inteligencia. En 3.7 se trata el tiempo y esfuerzo de programación y sus estimadores. En 3.8 se trata el nivel del lenguaje y su estimador.

3.2 Ciencia del Software

La Ciencia del Software [HALS77] establece que los algoritmos consisten en operadores y operandos, y trata con las propiedades de los algoritmos que pueden ser medidas, directa o indirectamente, estática o dinámicamente, así como con las relaciones entre aquellas propiedades que no cambian cuando se convierte de un lenguaje a otro.

Las relaciones que gobiernan la implementación de los algoritmos son: longitud, nivel, modularidad, pureza, volumen, contenido de inteligencia, el número de discriminaciones mentales y el tiempo requerido para escribirlo.

Las propiedades de un programa de computadora que se pueden contar o medir incluyen las siguientes métricas:

η_1 = Número de operadores diferentes. (1)

η_2 = Número de operandos diferentes. (2)

N_1 = Número total de operadores. (3)

N_2 = Número total de operandos. (4)

Los operadores son cualquier símbolo que represente una acción algorítmica, mientras que un símbolo utilizado para representar datos se considera un operando.

A partir de estas métricas básicas se define el vocabulario del programa (η), que consiste en el número de las diferentes partículas elementales usadas para construir un programa, como:

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

3.3 Longitud del Programa y su Estimador

La longitud de un programa (N), en términos del número total de partículas elementales usadas, se define como:

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

Para los programas escritos en lenguaje máquina, en los cuales cada línea de código (LOC) tiene un operador y un operando se tiene que N = 2*LOC.

La longitud del programa solamente es una función del número de operadores y operandos diferentes [SHEN83]. Esta es llamada ecuación de longitud y para poder estimar la longitud de un programa se establece la siguiente ecuación:

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \qquad (7)$$

Esta ecuación se debe considerar como una aproximación a la realidad.

Existe evidencia que sugiere la validez de la ecuación de longitud en varios lenguajes. La exactitud de la estimación depende de la longitud del programa, para programas del longitud grande ($N > 4000$) la ecuación subestima el valor de la longitud y para programas pequeños ($100 < N < 2000$) la ecuación sobrestima la longitud [SHEN83]; este comportamiento, el cual se ilustra en la figura 7 también se observó en el estudio realizado por Martínez [MART94].

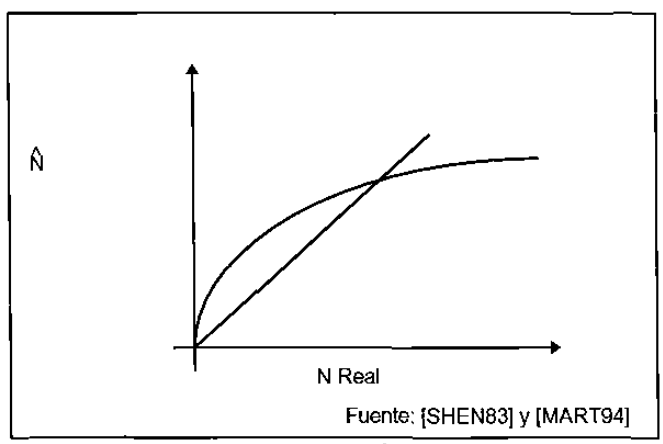


Figura 7 Comportamiento de $\hat{N}$.

3.4 Volumen del Programa y su Estimador

Una característica importante de un algoritmo es su tamaño. Los algoritmos tienen diferente tamaño cuando se implementan en diferentes lenguajes. Para

estudiar tales cambios en una forma cuantitativa se requiere que el tamaño sea una cantidad medible.

Una métrica para el tamaño de cualquier implementación de cualquier algoritmo, llamada volumen V, puede ser definida como:

$$V = N \log_2 \eta \qquad (8)$$

Esta interpretación da un volumen de programa con dimensión en bits. Si un algoritmo es convertido de un lenguaje a otro cambiará su volumen.

El volumen también se puede interpretar como el número de comparaciones mentales que se necesitan para escribir un programa de longitud N, suponiendo que se usa un método de inserción binaria para seleccionar un miembro del vocabulario de tamaño η.

La forma más breve en la cual un algoritmo pudiera ser expresado requeriría la existencia de un lenguaje en el cual la operación requerida ya estuviera definida o implementada, quizás como una subrutina o un procedimiento. En tal lenguaje, la implementación del algoritmo requeriría nada más el nombre de los operandos para sus argumentos y sus resultados. Ahora en esta forma mínima, ni los operadores ni los operandos podrían requerir repetición.

La siguiente ecuación denota el Volumen Potencial:

$$V^* = (\eta_1^* + \eta_2^*) \log_2 (\eta_1^* + \eta_2^*) \qquad (9)$$

El número mínimo posible de operadores η_1^* para cualquier algoritmo es conocido. Este debe consistir de un operador diferente para el nombre de la función o

procedimiento y otro para servir como una asignación o grupo de símbolos. Por lo tanto: $\eta_1^* = 2$.

Entonces la ecuación anterior se convierte en:

$$V^* = (2 + \eta_2^*) \log_2 (2 + \eta_2^*) \qquad (10)$$

donde η_2^* representa el número de los diferentes parámetros de entrada/salida. El Volumen Potencial de un algoritmo sería independiente de cualquier lenguaje en el cual pueda ser expresado.

3.5 Nivel / Dificultad del Programa y su Estimador

Cualquier programa con volumen V se considera que se implementa en el nivel del programa L, el cual se define por:

$$L = V^* / V \qquad (11)$$

El valor de L varía de 0 a 1, donde L = 1 representa un programa escrito en el más alto nivel posible.

Lo inverso del nivel del programa se llama dificultad del programa, y se define como:

$$D = 1 / L \qquad (12)$$

Cuando el volumen de una implementación de un programa crece, el nivel del programa decrece y la dificultad se incrementa. De este modo, las prácticas de programación tales como el uso redundante de operandos, o el error de usar frases de control de nivel más alto tenderán a incrementar el volumen así como la dificultad.

El nivel del programa depende de la razón entre el volumen potencial y el volumen actual. Ya que el volumen potencial frecuentemente no está disponible, una fórmula alternativa que estima el nivel se define como:

$$\hat{L} = (2 / \eta_1) (\eta_2 / N_2) \qquad (13)$$

El nivel del programa depende del lenguaje que está siendo utilizado, además, podría variar grandemente para programas equivalentes escritos en el mismo lenguaje porque este depende de la experiencia y el estilo del programador.

Los datos que muestran la validez de la ecuación del nivel del programa dependen de los valores de η_2^*, los cuales son determinados utilizando un método subjetivo. No se puede probar la ecuación del nivel objetivamente sobre programas grandes porque no se tiene un método objetivo para calcular η_2^*. También se demostró que D es una buena medida de la propensión al error [SHEN83].

El esfuerzo que se requiere para implementar un programa de computadora se incrementa cuando el tamaño del programa crece. También toma más esfuerzo implementar un programa en un nivel más bajo comparado con otro programa equivalente en un nivel más alto. El esfuerzo se define como:

$$E = V / L \qquad (14)$$

La unidad de medida de E es discriminaciones binarias mentales elementales [HALS77].

3.6 Contenido de Inteligencia

Cuando $\hat{L}$ se utiliza para estimar L, el producto $\hat{L}V$ es llamado contenido de inteligencia, y está definido por:

$$I = \hat{L} V \qquad (15)$$

El contenido de inteligencia es constante para diferentes implementaciones del mismo problema, ya que es un estimador de V*. En el estudio realizado por Shen [SHEN83] se pueden observar algunos problemas con respecto al contenido de inteligencia.

3.7 Esfuerzo y Tiempo de Programación y sus Estimadores

Stroud define "momento" como el tiempo requerido por el cerebro humano para desempeñar la discriminación más elemental. Estos momentos ocurren en un rango de cinco a veinte o un poco menos por segundo. Denotando los momentos de Stroud por segundo por S, tenemos: $5 \leq S \leq 20$ por segundo.

Para convertir la ecuación de E (14), la cual tiene dimensiones de dígitos binarios y discriminaciones, a unidades de tiempo, tenemos que dividir ambos lados por las discriminaciones por unidad de tiempo, obteniéndose:

$$T = E/S = V/SL = V^2/SV^* \qquad (16)$$

Esta ecuación puede ser expresada en términos de sus parámetros básicos:

$$\hat{T} = \eta_1 N_2 (\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2) \log_2 \eta / 2S\eta_2 \qquad (17)$$

Donde todos los parámetros de la derecha son medibles a excepción del número de Stroud S, el cual está normalmente colocado en 18, ya que esto pareció dar el mejor resultado en los experimentos de Halstead [HALS77].

En la derivación del tiempo de programación, Halstead confía en dos suposiciones. La primera es que el proceso de selección de un programador para

construir un programa (escogiendo operadores y operandos de un vocabulario (η)) lo aproxima a una búsqueda binaria.

La segunda suposición es que el humano es capaz de hacer un número constante (S) de discriminaciones mentales por segundo. Esto es cuestionable aunque uno puede aplicar esta hipótesis en esta situación. Deberíamos observar también que el trabajo de Stroud y el término del número de Stroud no están totalmente aceptados por los psicólogos debido a la falta de evidencia empírica. La presencia de estas y otras suposiciones no verificables en la derivación de las fórmulas arroja dudas serias en las fundaciones teóricas de la ciencia del software [SHEN83].

3.8 Nivel del Lenguaje y su Estimador

Halstead hipotetizó que si el lenguaje de programación se mantiene fijo, entonces mientras V* crece, L disminuye de tal forma que el producto L*V se mantiene constante. Así, este producto llamado nivel del lenguaje (λ), se puede usar para caracterizar un lenguaje de programación. Esto es:

$$\lambda = L * V^* = L^2V \tag{18}$$

Sustituyendo, $\hat{L}$ por L de la ecuación (13) y V de la ecuación (8), tenemos que el estimador para el nivel del lenguaje es:

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

Analizando un número de programas diferentes escritos en lenguajes diferentes, se determinaron los niveles de lenguaje para cada uno de ellos [HALS77] (ver tabla I).

En la tabla II se observan los valores obtenidos para los lenguajes de cuarta generación, FoxPro2 y DBaseIII, obtenidos en el el estudio realizado por Martínez [MART94].

Estos valores promedio obedecen a la mayoría de las clasificaciones intuitivas de los programadores para estos lenguajes, pero todos ellos tienen grandes varianzas. Tales fluctuaciones en un valor hipotetizado no son completamente inesperadas ya que el nivel del lenguaje no sólo depende del lenguaje en sí mismo, sino también de la naturaleza del problema que está siendo programado así como de la habilidad y estilo del programador [HALS77].

Tabla I

Media y Varianza del Nivel del Lenguaje.

Lenguaje	λ	σ^2
Inglés	2.16	0.74
PL/1	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: [HALS77]

Tabla II

Media y Desviación Estándar del Nivel del Lenguaje.

Lenguaje	λ	σ
DBaseIII	1.9544	1.7039
FoxPro2	1.9763	1.8112

Fuente: [MART94]

Esta métrica del nivel del lenguaje podría ser utilizada en la selección de un lenguaje para nuevas aplicaciones, en probar el poder potencial del lenguaje propuesto, y en la predicción del esfuerzo relativo para producir software equivalente en diferentes lenguajes de programación.

Existen diversos estudios del nivel del lenguaje en diferentes lenguajes. Estos estudios muestran una fuerte dependencia inversa en la longitud del programa. De estos resultados parece que el nivel del lenguaje tiene una función exponencialmente decreciente de la longitud del programa, destrozando la validez de la consistencia [SHEN83]. Este resultado también se observó en el estudio realizado por Martínez [MART94], tal como se muestra en la figura 8.

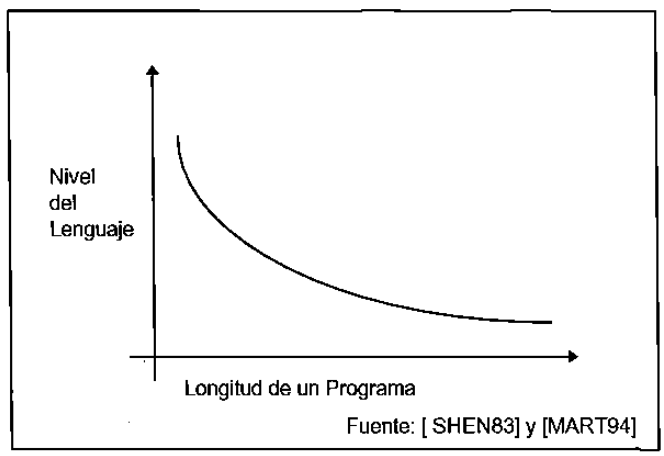


Figura 8 Comportamiento del Nivel del Lenguaje.

3.9 Resumen

En este capítulo se presentaron los fundamentos y algunas críticas de la Ciencia del Software propuesta por Halstead, así como algunos resultados adicionales obtenidos más recientemente.

CAPÍTULO 4

PARADIGMA DE ORIENTACIÓN A OBJETOS

4.1 Introducción

En este capítulo se presenta la teoría relacionada con el paradigma de la orientación a objetos. En 4.2 se muestra la historia, en 4.3 se menciona qué es la orientación a objetos, en 4.4 se mencionan ventajas, en 4.5 se mencionan los conceptos relacionados con el paradigma de la orientación a objetos. En 4.6 se trata el análisis y diseño orientados a objetos. En 4.7 se muestra una comparación entre el análisis y diseño convencional y el orientado a objetos. En 4.8 se menciona lo que es la programación orientada a objetos. En 4.9 se ve una comparación del método tradicional y el método orientado a objetos. En 4.10 se ven ejemplos de programación procedural estructurada y programación orientada a objetos. En 4.11 se trata el tema de las métricas de software orientadas a objetos. Y en 4.12 se muestran algunas áreas de aplicación

4.2 Historia del Paradigma de Orientación a Objetos

La orientación a objetos cambiará la forma de trabajar de los programadores y aumentará la velocidad de producción de la próxima generación de aplicaciones. También puede aumentar la capacidad de programación del usuario final [WINB93].

La orientación a objetos no es un concepto nuevo. Sus raíces pueden encontrarse en Noruega a finales de los años 60 en conexión con un lenguaje llamado Simula67, desarrollado por Kristen Nygaard y Ole-Johan Dahl en el Centro de Cálculo Noruego. Simula67 introdujo por primera vez los conceptos de clases, corrutinas y subclases, muy parecidos a los lenguajes orientados a objetos de hoy en día [WINB93].

En la mitad de la década de los 70´s, los científicos del Centro de Investigación de Palo Alto de Xerox (Xerox PARC) crearon el lenguaje Smalltalk, el primer lenguaje orientado a objetos consistente y completo [WINB93].

Existen dos ámbitos principales de los lenguajes orientados: uno es el grupo del lenguaje puro orientado a objetos, e incluye a Smalltalk, Actor de Whitewater Group y Eiffel de Interactive Software Engeering, Inc. El otro ámbito es el grupo híbrido, cuyas construcciones orientadas a objetos se añaden a un lenguaje procedimental. Los miembros de este grupo incluyen C++, Objective-C, Common Lisp Object System (CLOS) y los diferentes lenguajes Pascal orientado a objetos. La figura 9 indica la evolución de los dos grupos de lenguajes orientados a objetos [WINB93].

Hasta hace poco la orientación a objetos era lenta para penetrar la corriente principal de la comunidad de las computadoras. Esta migración era debida a que Simula67 y Smalltalk fueron inaccesibles a la corriente principal de la comunidad de computadoras hasta los años 80 [WINB93].

En los años 80, C se convirtió en un lenguaje de desarrollo muy popular, no sólo en las microcomputadoras sino en la mayoría de las arquitecturas y entornos de computación. Al principio de la década de los 80, Bjarne Stroustrup de los Laboratorios de AT&T amplió el lenguaje C para crear C++, un lenguaje que soporta la programación orientada a objetos. Posteriores mejoras en herramientas y

lanzamientos comerciales del lenguaje C++ por AT&T, y otros fabricantes, justifican buena parte de la atención general hacia la programación orientada a objetos en la comunidad de software. Con C++, los programadores eran capaces de aprender el paradigma de la orientación a objetos en un léxico popular y conocido, sin tener que invertir en nuevos y diferentes entornos y lenguaje de computación [WINB93].

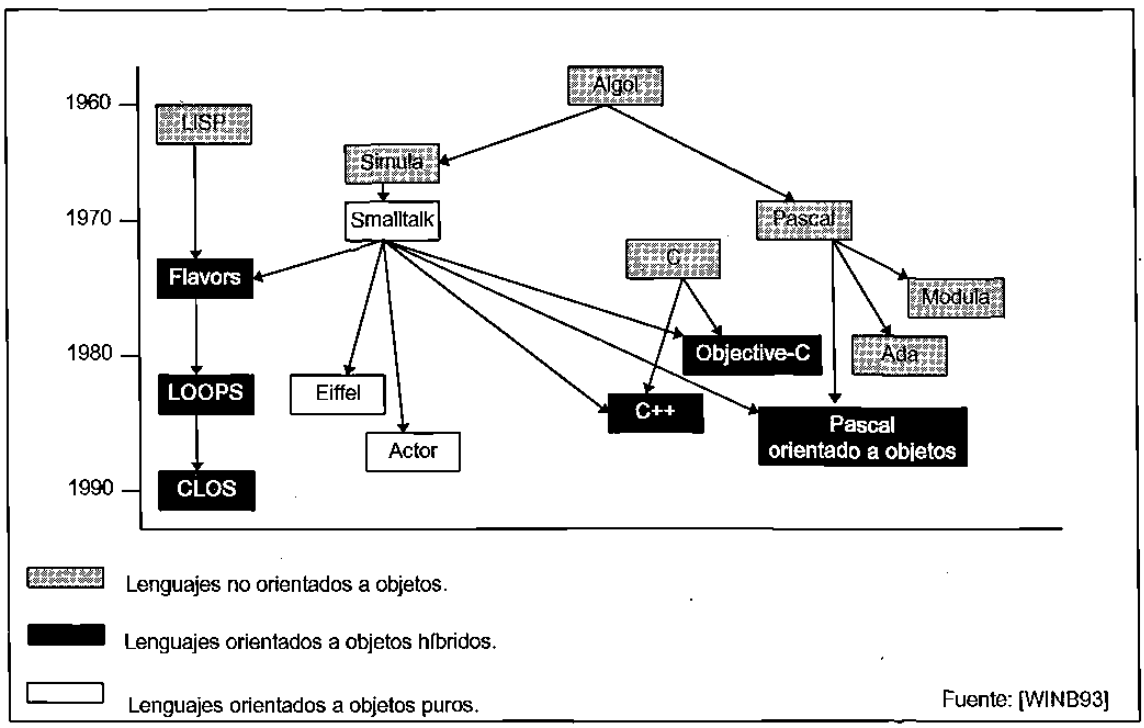


Figura 9 Evolución de los Lenguajes Orientados a Objetos.

La programación orientada a objetos proporciona una mejor forma de gestionar la complejidad tecnológica. La orientación a objetos permite la programación en niveles más altos de abstracción [WINB93].

4.3 ¿Qué es la Orientación a Objetos?

En la literatura existen diferentes definiciones de la orientación a objetos:

"Un sistema construido con métodos orientados a objetos es aquel cuyos componentes son datos y funciones encapsulados, las cuales pueden heredar

atributos y comportamientos a partir de otros componentes, y estos componentes se comunican entre ellos mediante mensajes" [YOUR94].

Según Khoshafian citado por Macías [MACI94], la orientación a objetos puede ser descrita como: "La modelación de software y disciplinas de desarrollo (ingeniería) que hacen fácil la construccion de sistemas complejos a través de componentes individuales". Además puede ser definida como sigue:

orientación a objetos = Tipos de datos abstractos

\+ Herencia

\+ Identidad de objetos.

El paradigma de la orientación a objetos, habla de un nuevo enfoque o una nueva manera de pensar acerca de los problemas, usando modelos organizados alrededor de conceptos del mundo real. El elemento fundamental en este nuevo enfoque es el objeto, el cual combina estructuras de datos y conducta en una entidad simple. Se destaca que la manera de analizar los problemas desde una perspectiva más cercana al mundo real, ayuda a que éstos sean expresados fácil y naturalmente [MACI94].

El atractivo de la orientación a objetos, es que ésta provee mejores conceptos y herramientas para modelar y representar el mundo real tan de cerca como sea posible [MACI94].

El desarrollo orientado a objetos es un enfoque para el diseño de software en el cual la descomposición de un sistema está basado en el concepto de objeto [BOOC86].

4.4 Ventajas del Paradigma de la Orientación a Objetos

Las principales ventajas descansan en su posibilidad de hacer frente a los dos temas esenciales de la ingeniería de software: gestión de la complejidad y mejora de la productividad en el proceso de desarrollo del software. La programación orientada a objetos conduce estos temas fomentando las siguientes estrategias de desarrollo del software [WINB93]:

- Escribir código reutilizable.
- Escribir código posible de mantener.
- Depurar módulos de código existentes.
- Compartir código con otros.

4.4.1 Gestión de la Complejidad

Descomponer una aplicación en entidades y relaciones que sean comprensibles para los usuarios es una técnica convencional de diseño y análisis. Con la programación orientada a objetos este proceso de descomposición se extiende también a la fase de realización [WINB93].

4.4.1.1 Flexibilidad en el Desarrollo del Software

Una vez que se han definido los objetos y se han ampliado las bibliotecas de clases, el proceso de programación puede facilitarse de forma creciente. El proceso de subclasificar por medio del mecanismo de herencia permite que la programación se convierta en un proceso de programar solamente las diferencias entre la subclase y superclase o la clase padre [WINB93].

4.4.1.2 Reutilización

Las técnicas orientadas a objetos ofrecen una alternativa para escribir los mismos programas una y otra vez. El programador orientado a objetos modifica la funcionalidad de un programa reemplazando los elementos u objetos antiguos por los nuevos objetos o simplemente incorporando nuevos objetos a la aplicación [WINB93].

4.4.2 Aumento de la Productividad

En esta sección se verán algunos aspectos relacionados con el aumento de la productividad.

4.4.2.1 Extensibilidad y Mantenibilidad

Es más fácil modificar y ampliar una aplicación orientada a objetos. Se pueden añadir nuevos tipos de objetos sin cambiar la estructura existente. La herencia permite construir nuevos objetos a partir de los antiguos. Los métodos son fáciles de modificar porque residen en una única localización, en lugar de estar dispersos y potencialmente repetidos a lo largo del programa [WINB93].

Las características de la orientación a objetos son extremadamente útiles durante el mantenimiento. La modularidad facilita el contener los efectos de los cambios realizados en un programa [WINB93].

4.4.2.2 Programación por el Usuario

El usuario de hoy en día escribe un número notorio de programas. Es probable que en el futuro, los usuarios que realizan programación orientada a objetos no reconozcan las tareas que ejecutan como programación por sí misma. Con la aplicaciones orientadas a objetos, las tareas de programación serán todavía más fáciles [WINB93].

En cuanto al usuario, el beneficio real de la orientación a objetos será el lanzamiento de bibliotecas de clases ricas en contenido que sean intuitivas en su representación del mundo real, así como fáciles de modificar y emplear [WINB93].

4.5 Conceptos del Paradigma de la Orientación a Objetos

En esta sección se verán los conceptos relacionados con el paradigma de la orientación a objetos.

4.5.1 Objeto

Como parte importante dentro de la metodología de orientación a objetos, está el entender lo que significa un objeto. En la literatura se encuentran las siguientes definiciones:

Un objeto es una abstracción de una entidad del mundo real [RINE92].

Los objetos son módulos que contienen los datos y las instrucciones que operan sobre esos datos, por lo tanto son entidades que tienen atributos (datos) y formas de comportamiento (procedimientos) particulares [WINB93].

Un objeto es una entidad que tiene estado, está caracterizado por las acciones que este sufre y por las que requiere de otros objetos, es una instancia de alguna clase, tiene un nombre, tiene visibilidad restringida de y por otros objetos y se puede ver su especificación y su implementación [BOOC86].

Un objeto es una abstracción de algo en el dominio del problema, reflejando las capacidades de un sistema para guardar información sobre él, o ambas; una encapsulación de valores de atributos y sus servicios exclusivos (sinónimo: instancia) [YOUR94].

Un objeto es una abstracción de alguna entidad dentro del dominio del problema que se está tratando, en la cual se encapsulan atributos (datos) y operaciones o servicios (conducta) [MACI94].

Un objeto es cualquier cosa que trate con el ambiente, tal como un carro, una computadora o una hamburguesa. Un objeto exhibe ciertos comportamientos. La orientación a objetos provee el mismo concepto en sistemas. En un sistema de información, los atributos (también llamados datos o estructuras de datos) y las operaciones están encapsuladas para crear objetos que se comportan de cierta manera [BURC92].

En conclusión un objeto es una entidad del mundo real la cual encapsula datos y operaciones.

4.5.2 Clase

Otro término dentro del paradigma de orientación a objetos que es necesario describir es el de clase, y algunas definiciones encontradas en la literatura son las siguientes:

Muchos objetos diferentes pueden actuar de formas muy similares. Una clase es una descripción de un conjunto de objetos casi idénticos. Una clase consta de métodos y datos que resumen las características comunes de un conjunto de objetos [WINB93].

Una clase es: una descripción de uno o más objetos con un conjunto uniforme de atributos y servicios, incluyendo una descripción de cómo crear nuevos objetos en la clase [YOUR94].

Una clase es un conjunto de objetos que comparten estructuras y comportamientos comunes [BURC92].

Una clase es un conjunto de o agrupamiento de objetos, los cuales comparten atributos y conductas similares. Una clase representa una abstracción de las características esenciales de un grupo de objetos [MACI94].

Una clase es un conjunto o colección de objetos que tienen características comunes [RINE92].

En conclusión una clase es un conjunto de objetos que tienen características y comportamientos comunes. Una clase es una abstracción de las principales características de un conjunto de objetos.

4.5.3 Herencia

Al igual que con los conceptos anteriores, se presentarán las definiciones que proponen algunos autores para la herencia:

La herencia es el mecanismo para compartir automáticamente métodos y datos entre clases, subclases y objetos; permite a los programadores crear nuevas clases programando solamente las diferencias con la clase padre [WINB93].

Herencia simple y múltiple son dos tipos de mecanismos de herencia. Con la herencia simple, una subclase puede heredar datos y métodos de una clase simple así como añadir o sustraer comportamiento por sí misma. La herencia múltiple se refiere a la posibilidad de una clase de adquirir los datos y métodos de más de una clase [WINB93].

La herencia es cualquier mecanismo que permite a un objeto incorporar todo o parte de la definición de otro objeto como parte de su propia definición [YOUR94].

La herencia permite compartir atributos y conductas de una o más clases previamente definidas, con una nueva clase; todo esto dentro de una jerarquía de clases. Las subclases pueden cambiar o agregar estructuras y conductas heredadas de la o las superclases (herencia simple y múltiple respectivamente) [MACI94].

La herencia es una relación entre dos clases de objetos, de tal manera que una de las clases, la hija, toma todas las características relevantes de la otra clase, la padre [RINE92].

La herencia es un mecanismo para la compartición de código o comportamiento comunes para un conjunto de clases [WEGN92].

En conclusión la herencia es un mecanismo para compartir atributos y operaciones definidas previamente en una clase.

4.5.4 Polimorfismo

En esta sección se presentan las definiciones de varios autores referentes al concepto de polimorfismo.

Como primer punto, se tiene que el significado general de polimorfismo se refiere a tener muchas partes o formas.

Los objetos actúan en respuesta a los mensajes que reciben. El mismo mensaje puede originar acciones completamente diferentes al ser recibido por diferentes objetos. Este fenómeno se conoce como polimorfismo. Con el polimorfismo un usuario puede enviar un mensaje genérico y dejar los detalles exactos de la realización para el objeto receptor [WINB93].

El polimorfismo está definido como un mecanismo que permite a operaciones diferentes en diferentes tipos de objetos tener el mismo nombre; como una

consecuencia práctica, esto significa que un objeto puede enviar un mensaje a otro objeto sin tener que conocer necesariamente la clase a la cual el objeto pertenece [YOUR94].

El polimorfismo es la propiedad en la cual, una misma operación puede tener un comportamiento distinto en clases diferentes [MACI94].

En conclusión el polimorfismo es un mecanismo que permite a una operación comportarse de manera diferente en diferentes clases.

4.5.5 Abstracción

El concepto de abstracción es utilizado ampliamente en el proceso de modelación. En esta sección se pretende describir el concepto de abstracción. Para ello, se utilizarán las definiciones que dan algunos autores.

La orientación a objetos fomenta que los programadores y usuarios piensen sobre las aplicaciones en términos abstractos. Comenzando con un conjunto de objetos, los programadores buscan un factor de comportamiento común y lo sitúan en superclases abstractas. Las bibliotecas de clases proporcionan un depósito para los elementos comunes y reutilizables [WINB93].

La abstracción es cualquier mecanismo que nos permita representar una realidad compleja en términos de un modelo simplificado [YOUR94].

La abstracción es un proceso mental que permite enfocarse en las características esenciales de un objeto, las cuales lo distinguen de los demás objetos e ignorar sus aspectos incidentales [MACI94].

La abstracción es la separación de detalles no necesarios de los requerimientos o especificación de sistemas para la reducción de la complejidad de entendimiento de los requerimientos o de especificación [RINE92].

En conclusión la abstracción es un mecanismo que nos permite representar una realidad compleja en un modelo simplificado, el cual reduzca la complejidad de entendimiento de los requerimientos o de especificación.

4.5.6 Mensajes y Métodos

A continuación se muestran algunas de las definiciones que dan algunos autores para los conceptos de mensaje y métodos.

Los objetos tienen la posibilidad de actuar. La acción sucede cuando un objeto recibe un mensaje, que es una solicitud que pide al objeto que se comporte de alguna forma. Los procedimientos llamados métodos residen en el objeto y determinan cómo actúa el objeto cuando recibe un mensaje [WINB93].

Un método es una operación que ya fue implantada en una clase, pero que puede ser redefinida en alguna subclase; el mensaje es la invocación de una operación y está constituido por el nombre de la operación y una lista de valores argumentos [MACI94].

Un método es una característica que ejecuta una operación sobre un objeto. Un mensaje es un protocolo compuesto de un método o rutina y referencias o direcciones de objetos [RINE92].

En conclusión un mensaje es un conducto por medio del cual se comunican los objetos y un método es una operación o procedimiento que se ejecuta sobre un objeto.

4.5.7 Encapsulación

Por último, se tiene el concepto de encapsulación. Para este concepto se presentan las definiciones dadas por diferentes autores.

La encapsulación es el término formal que describe el conjunto de métodos y datos dentro de un objeto de forma que el acceso a los datos se permite solamente a través de los propios métodos del objeto. Ninguna otra parte de un programa orientado a objetos puede operar directamente sobre los datos de un objeto [WINB93].

La encapsulación es cualquier mecanismo que nos permite esconder la implementación de un objeto, por eso otros componentes del sistema no estarán conscientes de los datos internos almacenados en el objeto [YOUR94].

La encapsulación u ocultamiento de información, consiste en mostrar sólo los aspectos externos del objeto (parte pública), aquellos que son accesibles a otros objetos y ocultar los detalles de implantación, aspectos internos del objeto (parte privada) [MACI94].

En conclusión la encapsulación es el término que describe a los datos y métodos dentro de un objeto, de manera que el acceso a los datos del objeto sea a través de los métodos del objeto.

4.6 Análisis / Diseño Orientado a Objetos

Algunas descripciones para el análisis y diseño encontradas en la literatura son las siguientes:

El análisis orientado a objetos es el análisis de las necesidades de un sistema expresado en términos de objetos del mundo real [WINB93].

Según Sally Shlaer y Stephen citado por Winbald [WINB93] el análisis comienza por definir los objetos y atributos. A continuación se definen los ciclos de vida de los objetos en modelos de estado para capturar los sucesos que actúan sobre objetos. El último paso es la definición de procesos, basada en los objetos y sus ciclos de vida.

Para Yourdon el análisis orientado a objetos consiste de las siguientes actividades [YOUR94]:

- Descubrir los objetos.
- Identificar las estructuras de objetos y la jerarquía de las clases.
- Identificar las relaciones de los objetos.
- La definición de atributos.
- Determinar el comportamiento del objeto.
- La definición de servicios.

El análisis orientado a objetos es una actividad que pretende modelar los sistemas del mundo real, para ello se realiza una búsqueda de objetos, clases, atributos y comportamientos. La parte esencial consistirá en la modelación de *qué* debe ser realizado por el sistema; y no *cómo* esto deberá ser realizado [MACI94].

El diseño orientado a objetos es la traducción de la estructura lógica de un sistema a una estructura física compuesta de objetos de software [WINB93].

El diseño orientado a objetos (DOO) se puede definir como una técnica que propone hacer una descomposición de un sistema en clases y objetos (diseño lógico); y en módulos y procesos (diseño físico). El fin del DOO es crear un representación del

dominio del problema que pueda ser transformada en software; para ello, se toma en cuenta *cómo* será resuelto el problema [MACI94].

El proceso de desarrollo orientado a objetos definido por Booch [BOOC86] tiene los siguientes pasos principales:

- Identificar los objetos y sus relaciones.
- Identificar las operaciones que afectan y requieren los objetos.
- Establecer la visibilidad de cada objeto en relación a otros objetos.
- Establecer la interface de cada objeto.
- Implementar cada objeto.

4.7 Comparación de las Metodologías de Análisis y Diseño Convencionales y Orientadas a Objetos

Las metodologías de análisis convencionales y las orientadas a objetos pueden ser comparadas en 11 aspectos, estos aspectos representan un superconjunto de aspectos soportados por las metodologías individuales (ver tabla III). La tabla muestra las similitudes y diferencias entre las metodologías [FICH92].

La diferencia más importante entre las metodologías de análisis convencionales y orientadas a objetos, es que los requerimientos de estas últimas tienen las operaciones encapsuladas. Las metodologías convencionales proveen herramientas para crear una descomposición funcional de las operaciones (renglón 8) y para modelar secuencias de procesos punto a punto (renglón 9). Una descomposición funcional viola la encapsulación, porque las operaciones pueden ser accesadas

directamente por una gran cantidad de entidades diferentes y no están subordinadas a ninguna otra entidad [FICH92].

Tabla III

Comparación de Metodologías de Análisis.

Componente	Análisis Estructurado de DeMarco	Análisis Estructurado Moderno de Yourdon	Ingeniería de la Información de Martin	Especificación de Requerimientos Orientado a Objetos de Bailin	Análisis Orientado a Objetos de Coad y Yourdon	Análisis Orientado a Objetos de Shlaer y Mellor
1. Identificación/ clasificación de entidades	No soportado	Diagrama de entidad - relación	Diagrama de modelo de datos	Diagrama de entidad - relación	Diagrama de clases y objetos capa 1	Diagrama de estructura de información
2. Relaciones de entidades de lo general a lo específico y de todo a parte	No soportado	Diagrama de entidad - relación	Diagrama de modelo de datos	Diagrama de entidad - relación	Diagrama de clases y objetos capa 2	Diagrama de estructura de información
3. Otras relaciones de entidades	No soportado	Diagrama de entidad - relación	Diagrama de modelo de datos	Diagrama de entidad - relación	Diagrama de clases y objetos capa 4	Diagrama de estructura de información
4. Atributos de entidades	Diccionario de datos	Diccionario de datos	Gráfica de burbujas	No soportado	Diagrama de clases y objetos capa 4	Diagrama de estructura de información
5. Particionamiento de un modelo grande	Diagrama de flujo de datos	Diagrama de flujo de datos de evento - particionado	Materia de bases de datos	Diagramas de entidad relación de dominio - particionado	Diagrama de clases y objetos capa 3	Gráficas de dominio, comunicación de subsistemas, accesos, y modelos de relaciones
6. Estados y transiciones	No soportado	Diagrama de transición de estados	No soportado	No soportado	Diagrama de estados de objetos, gráfica de servicios	Modelo de estados
7. Lógica detallada para servicios / funciones	Mini - especificación	Mini - especificación	No soportado	No soportado	Gráfica de servicios	Diagrama de acciones de flujo de datos, descripciones de procesos
8. Descomposición arriba-abajo de funciones	Diagrama de flujo de datos	Diagrama de flujo de datos	Diagrama de descomposición de procesos	No soportado	No soportado	No soportado

Tabla III (Continuación)

Componente	Análisis Estructurado de DeMarco	Análisis Estructurado Moderno de Yourdon	Ingeniería de la Información de Martin	Especificación de Requerimientos Orientado a Objetos de Bailin	Análisis Orientado a Objetos de Coad y Yourdon	Análisis Orientado a Objetos de Shlaer y Mellor
9. Secuencias de procesos punto a punto	Diagrama de flujo de datos	Diagrama de flujo de datos	Diagrama de dependencias de procesos	No soportado	No soportado	No soportado
10. Identificación de servicios exclusivos	No soportado	No soportado	No soportado	Diagrama de entidades - flujo de datos	Diagrama de clases y objetos capa 5	Modelo de estados, diagrama de acciones de flujo de datos
11. Comunicación de entidades	No soportado	No soportado	No soportado	Diagrama de entidades - flujo de datos	Diagrama de clases y objetos capa 5	Modelos de comunicación de objetos, modelo de acceso - objetos

Fuente: [FICH92]

Las distinciones entre el desarrollo convencional y el orientado a objetos son ampliadas durante el diseño debido a la importancia de aspectos específicos de la implantación (ver tabla IV). Ninguna de las metodologías convencionales soportan la definición de clases, herencia, métodos, o protocolos de mensajes. Ambas metodologías proveen herramientas que definen una jerarquía de módulos, se emplea un método completamente diferente, y la definición del término módulo es muy diferente [FICH92].

En los sistemas convencionales, los módulos sólo contienen código procedural. En los sistemas orientados a objetos la unidad principal de modularidad es el objeto. Las metodologías convencionales emplean una descomposición orientada a la función y las metodologías orientadas a objetos emplean una descomposición orientada a los objetos [FICH92].

Tabla IV

Comparación de Metodologías de Diseño.

Componente	Diseño Estructurado de Yourdon y Constantine	Ingeniería de la Información de Martin	Diseño Orientado a Objetos de Wasserman et al.	Diseño Orientado a Objetos de Booch	Diseño de Manejo - Responsabilidad de Wirfs - Brock et al.
1. Jerarquía de módulos	Gráfica de estructura	Diagrama de descomposición de procesos	Gráfica de estructura orientada a objetos	Diagrama de módulos	No soportada
2. Definiciones de datos	Diagrama de jerarquía	Diagrama de modelo de datos, diagrama de estructura de datos	Gráfica de estructura orientada a objetos	Diagrama de clases	Especificación de clases
3. Lógica procedural	No soportada	Diagrama de acciones	No soportada	Plantilla de operaciones	Especificación de clases
4. Secuencias de procesos punto a punto	Diagrama de flujo de datos	Diagrama de flujo de datos, diagrama de dependencia - procesos	No soportada	Diagramas de tiempo	No soportada
5. Estados de objetos y transiciones	No soportada	No soportada	No soportada	Diagrama de transición de estados	No soportada
6. Definición de clases y herencia	No soportada	No soportada	Gráfica de estructura orientada a objetos	Diagrama de clases	Diagrama de jerarquía
7. Otras relaciones	No soportada	No soportada	Gráfica de estructura orientada a objetos	Diagrama de clases	Especificación de clases
8. Asignación de servicios/ operaciones a clases	No soportada	No soportada	Gráfica de estructura orientada a objetos	Diagrama de clases	Gráfica de colaboraciones, especificación de clases
9. Definición detallada de operaciones / servicios	No soportada	No soportada	No soportada	Plantilla de operaciones	Especificación de clases
10. Conexiones de mensajes	No soportada	No soportada	Gráfica de estructura orientada a objetos	Diagrama y Plantilla de objetos	Gráfica de colaboraciones

Fuente: [FICH92]

4.8 Programación Orientada a Objetos

En la literatura se encuentran las definiciones de algunos autores que se muestran a continuación, pero más que dar una definición, se citan las características que la describen.

La programación orientada a objetos es una metodología para crear programas utilizando conjuntos de objetos auto-suficientes que tienen datos y comportamiento encapsulados, actuando a petición, e interactuando con cada uno de ellos mediante el paso de mensajes en ambos sentidos [WINB93].

La programación orientada a objetos es un método de desarrollo orientado a objetos que conduce a sistemas de software basados en los objetos que cada sistema / subsistema manipula, en vez de la función que estos pretenden asegurar [RINE92].

Según Booch citado por Brooks [BROO94] la programación orientada a objetos es un método de implementación en la cual los programas son organizados como conjuntos de objetos cooperativos, cada uno de los cuales representa una instancia de alguna clase, y en esas clases están todos los miembros de una jerarquía de clases unidos por relaciones de herencia.

La programación orientada a objetos es un nuevo paradigma que propone la implantación de programas como colecciones cooperativas de objetos, haciendo énfasis en el empleo de clases, herencia, polimorfismo, comunicación entre objetos mediante mensajes y encapsulamiento. Cabe hacer notar que si la programación sólo se centra en el uso de objetos no puede ser llamada orientada a objetos, sino basada en ellos. Las características deseables que debería soportar la POO serían: objeto, clase, herencia, polimorfismo, comunicación con mensajes y encapsulamiento [MACI94].

En conclusión la programación orientada a objetos es una metodología para crear programas como colecciones cooperativas de objetos, tomando en cuenta los conceptos de objetos, clase, herencia, polimorfismo, comunicación entre objetos y encapsulamiento.

4.9 El Método de Programación Tradicional Frente a la Programación Orientada a Objetos

Desde el punto de vista de un programador algunas técnicas orientadas a objetos parecen ser conceptos tradicionales con nombres diferentes. Algunos conceptos orientados a objetos son análogos a los métodos de programación convencional. La tabla V muestra algunos contrastes entre términos y conceptos convencionales, y aquellos orientados a objetos [WINB93].

Tabla V

Comparación de Conceptos de Programación Tradicional y la Orientada a Objetos.

Técnicas orientadas a objetos	Técnicas tradicionales
Métodos	Procedimientos, funciones o subrutinas
Variables modelo	Datos
Mensajes	Llamadas a procedimientos o funciones
Clases	Tipos abstractos de datos
Herencia	(No existe técnica similar)
Llamadas bajo control del sistema	Llamadas bajo control del programador

Fuente: [WINB93]

Otra comparación entre la programación tradicional y la orientada a objetos obtenida por Khan [KHAN95] se muestra en la tabla VI.

Tabla VI

Comparación de Características de Programación Procedural Estructurada y la Programación Orientada a Objetos.

Programación procedural estructurada	Programación orientada a objetos
1. Los sistemas son modularizados en base a sus funciones.	Los sistemas son modularizados en base a sus estructuras de datos (objetos).
2. En un módulo de programa, los datos y los procedimientos están separados.	En un módulo de programa, el estado de los objetos (tipos de datos) y el comportamiento (operaciones) están encapsuladas.
3. Los programadores son responsables de las llamadas de los procedimientos activos para el paso de parámetros.	Los objetos activos se comunican con otro objeto por el paso de mensajes para activar sus operaciones.

Tabla VI (Continuación)

Programación procedural estructurada	Programación orientada a objetos
4. Los usuarios se deben de asegurar que el procedimiento trabajará correctamente sobre el tipo de datos en el cual se esta aplicando.	El paso de mensajes asegura que el estado interno del objeto puede ser accesado sólo si es permitido, la encapsulación previene el acceso no autorizado.
5. El mundo real esta representado por las entidades lógicas y el flujo de control.	El mundo real esta representado más cercanamente por los objetos imitando las entidades externas.
6. Los módulos de programa están enlazados a través del mecanismo de paso de parámetros y el sistema operativo.	Los módulos de programas son partes integradas del sistema general, un programa es una colección de objetos interactuando.
7. Utiliza abstracción procedural.	Utiliza abstracción de clases y objetos.
8. Los métodos (operaciones) utilizan estructuras de datos pasivas y tontas.	Las estructuras de datos (objetos) activos e inteligentes encapsulan todos los procedimientos pasivos.
9. Unidad de estructura: línea o expresión.	Unidad de estructura: el objeto tratado como un componente de software.
10. Utiliza descomposición funcional.	Utiliza descomposición orientada a objetos.
11. Utiliza lenguajes orientados a procedimientos tales como C o Pascal.	Utiliza lenguajes orientados a objetos tales como C++, Object Pascal, o Smalltalk-80.

Fuente [KHAN95]

4.9.1 Beneficios de la Programación Orientada a Objetos

Los siguientes beneficios son específicos de la programación orientada a objetos (POO) y no pueden ser obtenidas utilizando técnicas de programación procedural estructurada [KHAN95].

- La POO provee una mejor estructura de sistema en la modelación del mundo real mejor.

- El concepto de abstracción de datos, junto con los principios de encapsulación y ocultamiento de información, incrementa la confiabilidad y modificabilidad por la implementación del objeto.

- El polimorfismo y el enlace dinámico incrementan la flexibilidad y la reutilización del código, permitiendo la creación de componentes de software genéricos y clases nuevas utilizando código existente.

- La herencia permite la extensibilidad y la reutilización del código de software, porque se pueden agregar atributos nuevos y operaciones nuevas (o se pueden borrar las anteriores) a través de la creación de clases hijas nuevas sin tener que modificar el código original.

- La POO facilita la construcción de prototipos y enfoque interactivo para el desarrollo de software más cercanamente que el ciclo de vida.

- La POO permite la utilización de librerías de componentes de software reutilizables para construir módulos de nuevas aplicaciones de software.

4.10 Ejemplos de Programación Procedural Estructurada (PPE) y Programación Orientada a Objetos (POO) Utilizando C++

La figura 10 muestra el listado completo de tres módulos de un programa estructurado para resolver un problema simple de cuentas bancarias utilizando algunas de las características de C++ para desarrollar programas estructurados. El módulo 1 contiene la declaración de la estructura de datos *cuenta* y el prototipo de las funciones compute_intfn(), depositfn(), y withdrawfn(). El módulo 2 contiene el programa principal, el cual crea dos variables de la estructura *cuenta*: *acct1* y *acct2* y utiliza las tres funciones declaradas en el módulo 1 para desempeñar los cálculos requeridos. Finalmente el módulo 3 contiene las definiciones de todas las funciones declaradas [KHAN95].

```
//file bkspp2.cpp
#include <iostream>
//module 1: declaration of struct and function
    prototype
    struct account
    { float balance;
    float interestrate;}; //declaration of struct
float depositfn (account& acctx, float newamount);
float withdrawfn (account& acctx, float newamount);
void compute_intfn (account& acctx);

//module 2: body of main program module which uses
    the struct and function declaration to
      create savings accounts
main()
{
    account acct1 = {500.50, 0.00065}; // acct1 and
    acct2 data objects
    account acct2 = {200.75, 0.0075}; // initialized
    with balance and rate
//print initial balance and monthly interest rate of
    acct1 and acct2
cout << "initial acct1 balance = BD" << acct1.balance
    << endl;
cout << "initial monthly interest rate = BD" <<
    acct1.interestrate << endl;
cout << "initial acct2 balance = BD" << acct2.balance
    << endl;
cout << "initial monthly interest rate = BD" <<
    acct2.interestrate << endl;
cout << endl;
//get user amount to deposit into account acct1
float userdeposit;
cout << "Please type amount to be deposited into
    acct1:";
cin >> userdeposit;
//update and print acct1 balance after deposit
deposit(acct1,userdeposit); //function call with two
    arguments
cout << "amount deposited into acct1 = BD"
<< depositfn(acct1,userdeposit) << endl;
cout << "acct1 new balance after deposit = BD" <<
    acct1.balance << endl;
cout << endl; //get user amount to withdraw from
    acct1
float userwith;
cout << "please type amount to withdraw from
    acct1:";
cin >> userwith;
//update and print acct1 balance and withdrawn
    amount
cout << "amount withdrawn from acct1 = BD" <<
    withdrafn(acct1,userwith) << endl;
cout << "acct1 new balance after withdrawal = BD"
    << acct1.balance << endl;
//get new monthly interes rate decided by the bank
    for acct1
float bankrate;
cout << "please type the new interest rate for acct1:";
cin >> bakrate;
acct1.interestrate = bankrate; //directly updated
cout << "acct1 new monthly interest rate = BD" <<
    acct1.interestrate << endl;
cout << endl;
//compute monthly interest, update and print new
    balance for acct1
compute_intfn(acct1);
cout << "acc1 new balance after another month =
    BD" <<acct1.balance << endl;
cout << endl;
} //end of main program

//module 3: definition of functios used in main()
float depositfn(account& acctx, float newamount)
{acctx.balance += newamount;
return newamount; } //newamount value returned
float withdrawfn(account& acctx, float newamount)
{
if (newamount <= acctx.balance)
    {
    acctx.balance -= newamunt;
    return newamount;
    }
else
    return 0.00; //balance smaller, no withdrawal
    allowed
} // end of withdrawfn
void compute_intfn(account& acctx)
    {float profit = acctx.balance * acctx.interestrate;
    acctx.balance += profit;
    return; }// end of compute_intfn
```

Fuente: [KHAN95]

Figura 10 Listado de un Programa Procedural Estructurado para Cuentas de Ahorro.

La figura 11 muestra un listado completo de los módulos de los programas orientados a objetos para el problema de cuentas bancarias. Aquí se crea una clase objeto llamada csavings_account. Al igual que en el programa estructurado, el programa orientado a objetos (OO) consta de tres módulos. El módulo 1 contiene la declaración de los miembros de datos y las funciones miembros de la clase csavings_account. El módulo 2 contiene las definiciones de todas las funciones

miembro de la clase declarada. Finalmente, el módulo 3 contiene la función principal. Ésta utiliza la declaración y definición de la clase para construir las instancias de la misma como objetos para desempeñar los cálculos requeridos [KHAN95].

```
//file boop.cpp
#include <iostream>
//module 1: declaration of class csavings_account
    class csavings_account
{
public:  //declaration of member funtions
csavings_account (float initbalance, float initrate); //
    constructor
~csavings_account(); //destrcutor
float get_balancefn(); // get balance for object
    account
float get_intratefn(float newrate); //set new interest
    rate
float depositfn(float new amount); // amount
    deposited
float withdrawfn(float newamount); amount withdrawn
void compute_intfn(); //compute interest and add to
    balance
private:  //declaration of data members
float balance; //opening balance amount
float interestrate; //opening interest rate
}; //end of declaration of class csavings_account

module 2:defnition of member functions
csavings_account::csaving_account(float initbalance,
    float initrate)
{balance = initbalace;
interestrate = initrate;} // no return statement
    requiered
csavings_account::~csavingsaccount()
{ cout << "savings_account with balance = BD" <<
    balance << endl;
cout << "and interest rate = BD"<< interesrate <<
    "dostroyed" << endl;
} //no return type required
float csavings_account::get_balancefn()
{ return balance;} // return current balance
float csavings_account::get_intratefn
{ return interesrate;} //return current interest rate
viod csavings_account::set_intratefn(float newrate)
{ interestrate = newrate; } //return current interest rate
float csavings_account::depositfn(float newamount)
{balance += newamount;
return newamount; } //return new amount deposited
float csavings_account::withdrawfn(float newamount)
{if(newamount <= balance)
{
balance -= newamount;
return newamount;
}
else
return 0.00; //balance smaller, allow no withdrawal
} // end of withdrawfn
void csavings_account::compute_intfn()
{float profit = balance*interestrate;
balance += profit;
return;} //end of compute_intfn
//end of module 2 definition of 8 functions

//module 3: main program module which uses the
    class csavings_account
main()
{
//create two instances (objects) of class
    savings_account
csavings_account acct1 (550.50, 0.0065);
csavings_account acct2 (750.75, 0.0075);
// print the initial balance and interest rate for objects
    acct1 & acct2
cout << "for acct1 initial balance = BD" <<
    acct1.get_balancefn() << endl;
cout << "for acct1 initial monthly interest rate = BD"
    << acct1.get_intratefn() << endl;
cout << endl;
cout << "for acct2 initial balance = BD" <<
    acct2.get_balancefn() << endl;
cout << "for acct2 initial monthly interest rate = BD"
    << acct2.get_intratefn() << endl;
cout << endl;
// get user amount for deposit into object acct1
float userdeposit;
cout << "please type the amount to be deposited into
    acct1:";
cin >> userdeposit;
//update and print the acct1 object balance after
    deposit
cout<< "amount deposited into acct1 = BD" <<
    acct1.depositfn(userdeposit) << endl;
cout << "acct1 balance after deposit = BD" <<
    acct1.get_balancefn() << endl;
cout << endl;
// compute the monthly interest, update and print new
    balance for object 1
acct1.computeintfn(); // object calls a method
    (operation)
cout << "for acct1 balance with interest after after one
    month = BD << acct1.get_balancefn() << endl;
cout << endl;
//get user amount to withdraw from acct1
float userwith;
cout << "please type the amount to be withdrawn
    form acct1:";
cin >> userwith;
//update and print the object acct1 balance after
    withdrwal
cout << "amount withdrawn form acct1 = BD" <<
    acct1.withdrawfn(userwith) << endl;
cout << "acct1 balance after withdrawl = BD" <<
    acct1.get_balancefn() << endl;
cout << endl;
// get new monthly interest rate decided by the bank
    for savings account
```

Figura 11 Programa Orientado a Objetos para Cuentas de Ahorro.

```
float bankrate ;
cout << "please type the new interest rate to be used
    for acct1:";
cin >> bankrate;
//update and print the new interest rate
acct1.setintratefn(bankrate);
cout << "for acct1new monthly interest rate = BD" <<
    acct1.get_intratefn() << endl;
cout << endl;
//compute the monthly interest, update and print new
    balance for object 1
acct1.compute_intfn(); //object1 calls a method
    operation)
cout << "for acct1 new balance with balance with
    interest after another month = BD" <<
    acct1.get_balancefn() << endl;
cout << endl;
} //end of program main
```

Fuente: [KHAN95]

Figura 11 (Continuación)

La figura 12 muestra gráficamente el contraste en las interacciones de los componentes del programa procedural estructurado y los objetos del programa orientado a objetos para el problema de las cuentas bancarias [KHAN95].

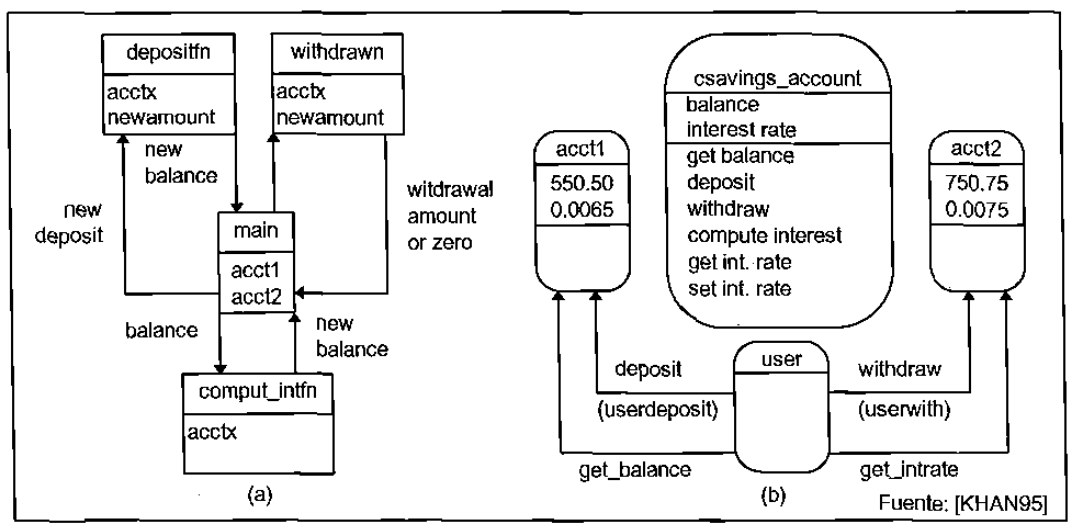


Figura 12 Representación Gráfica de los Programas Procedural Estructurado (a) y Orientado a Objetos (b) para el Problema de Cuentas Bancarias.

La comparación de los dos programas revela los siguientes puntos [KHAN95]:

- Los miembros de datos y las funciones miembro son encapsuladas en la POO pero no en la PPE.

- Las definiciones de las funciones en la PPE y la POO son similares, pero son más simples en el último caso.

- La inicialización en la PPE es simple pero no provee protección para el acceso no autorizado a las variables. En la POO necesitamos la función especial constructor para la inicialización de protección contra el acceso no autorizado a los miembros privados.

- La inicialización automática por la función constructor asegura que el estado del objeto no reflejará basura. Similarmente, una función destructor especial ejecuta una limpieza después de que el objeto deja de existir.

- El acceso a la variable *balance* en la PPE es simple porque no existe protección, en la POO se necesita una función miembro adicional para el acceso de los miembros privados, llamada función *get_balance()*.

- Una llamada de función en la PPE pasa y recibe datos como argumentos, mientras que en la POO los objetos invocan directamente a los métodos.

Una observación importante de las funciones en la PPE y la POO muestra que en la primera las funciones son entidades activas y pasan los datos pasivos *acct1* y *acct2*, mientras en el segundo caso los objetos activos *acct1* y *acct2* activan las funciones pasivas como métodos. Por ejemplo, en la POO tenemos *acct1.depositfn(userdeposit)*, donde el objeto activo *acct1* llama a la función pasiva *depositfn* para actualizar el valor del depósito del usuario, mientras que en la PPE tenemos *depositfn(acct1,userdeposit)*, donde la función activa *depositfn* pasa los datos pasivos *acct1* para que sean actualizados por el valor del depósito del usuario *userdeposit*. Se puede concluir que la PPE y la POO tratan con los mismos componentes: estructuras de datos (objetos) y funciones (operaciones). La PPE identifica las funciones del sistema que se van a desarrollar, y cada función opera sobre alguna estructura de datos, mientras que la POO identifica los objetos que constituyen el sistema, y cada objeto invoca algunas funciones [KHAN95].

4.11 Métricas de Software Orientadas a Objetos

Yourdon sugiere que la madurez del proceso es la introducción de métricas que van hacia el mejoramiento del proceso. Es difícil de imaginar una organización que haga cualquier mejora a su productividad o calidad si no mide lo que está haciendo. La motivación fundamental de las métricas de software es bastante simple: el deseo de mejorar [YOUR94].

En el área del paradigma de orientación a objetos Chidamber ha desarrollado un conjunto de métricas para mejorar el proceso del diseño orientado a objetos [CHID94]. Laranjeira presenta un modelo para la estimación del tamaño de sistemas orientados a objetos [LARA90] y Tegarden utiliza las métricas tradicionales, tales como las líneas de código, la complejidad ciclomática de McCabe y la ciencia del software de Halstead para la medición de la complejidad de los sistemas orientados a objetos, con algunas restricciones de no poder medir el grado de herencia y el polimorfismo, entre otras características del paradigma de la orientación a objetos [TEGA92]. Brooks y Buell desarrollaron una herramienta automática para la aplicación de métricas de software orientadas a objetos, tales como: a nivel de clase tenemos, la profundidad del árbol de la herencia, acoplamiento de clases, respuesta para una clase, y número de hijos; a nivel de sistema está el número de jerarquías de clase [BRRO94].

4.12 Áreas de Aplicación

El paradigma de la orientación a objetos se puede utilizar en las aplicaciones de sistemas de bases de datos, sistemas expertos e interfaces orientadas a los objetos [PRES93]. Otras áreas de aplicación son: (1) generación de interfaces gráficas para el usuario, (2) desarrollo de sistemas bancarios, (3) desarrollo de sistemas financieros, (4) programas de comunicación, (5) desarrollo de objetos activos tales como agentes.

Las aplicaciones de un entorno orientado a objetos serán transparentes. Las estructuras orientadas a objetos facilitarán la simulación y la construcción de soluciones específicas para el usuario. Los objetos serán compartidos en entornos de red para distribuir información dentro de un grupo de trabajo o para repartir las tareas de un procesamiento distribuido [WINB93].

En resumen las aplicaciones orientadas a objetos tienen tres ventajas principales [WINB93]:

- Mayor flexibilidad: permite adaptar y conformar la funcionalidad a las necesidades y preferencias de cada persona.

- Integración transparente: proporciona a los usuarios integración de información en directo, mientras que, al mismo tiempo, facilitarán la circulación de los datos entre aplicaciones y la posibilidad de que todos las compartan.

- Empleo más fácil: quizá la mayor ventaja de estas aplicaciones sea la facilidad de utilización proporcionada al usuario final por la mayor similitud entre el problema en cuestión y la solución que desarrolla la computadora.

4.13 Resumen

En este capítulo se presentó el paradigma de la orientación a objetos, la evolución de los lenguajes orientados a objetos, los conceptos básicos, también se muestra una comparación de las metodologías de análisis y diseño convencionales y orientadas a objetos, además de otra comparación de programación estructurada y la programación orientada a objetos.

CAPÍTULO 5

METODOLOGÍA

5.1 Introducción

En este capítulo se presentan las preguntas de la investigación y la metodología a seguir. En 5.2 se presentan las preguntas de la investigación. En 5.3 se presenta la metodogía utilizada. En 5.3.1 se presenta el diseño de la investigación. En 5.3.2 se presenta la selección del lenguaje de programación orientado a objetos. En 5.3.3 se presenta la selección de la muestra y en 5.3.4 se habla acerca del analizador de código.

5.2 Preguntas de la Investigación

Existe evidencia empírica de que el estimador de la longitud de un programa ($\hat{N}$) es un buen estimador de la longitud del programa (N) para lenguajes de tercera [SHEN83] y cuarta generación [MART94] desde el punto de vista estadístico. En consecuencia, podemos proponer nuestra primer pregunta de investigación: ¿Es el estimador $\hat{N}$ un buen estimador de N, en lenguajes de programación orientados a objetos? Con esta pregunta, formulamos nuestra primer hipótesis de investigación:

H_1: Para lenguajes de programación orientados a objetos, $\hat{N} = N$.

Pressman [PRES93] ubica a los lenguajes orientados a objetos dentro de la tercera generación de lenguajes, esta generación se divide en tres categorías, las cuales son lenguajes de alto nivel de propósito general, lenguajes de alto nivel orientados a objetos y lenguajes especializados. Por esta razón se espera que el nivel del lenguaje de los lenguajes de programación orientados a objetos (LPOO) sea mayor que el nivel del lenguaje de los lenguajes de tercera generación (3GL's) que utilizó Halstead en su estudio [HALS77], (los cuales están ubicados en la categoría de lenguajes de alto nivel de propósito general) y menor que el nivel del lenguaje de los lenguajes de cuarta generación (4GL's).

Esto nos conduce a la siguiente pregunta de investigación: ¿Es el nivel del lenguaje de los LPOO mayor que el nivel del lenguaje de los 3GL's y menor el nivel del lenguaje de los 4GL's? Con esta pregunta, formulamos las siguientes hipótesis de investigación:

H_2: Para lenguajes de programación orientados a objetos, el nivel del lenguaje es mayor que 1.53 y menor que 1.9763.

H_3: Para lenguajes de programación orientados a objetos, el nivel del lenguaje es mayor que 1.53 y menor que 1.9544.

5.3 Metodología

En esta sección se muestra la selección del diseño de la investigación. También se muestra la selección del LPOO y la selección de la muestra.

5.3.1 Diseño de la Investigación

La investigación se desarrolló como un estudio expost-facto utilizando un analizador de código para programas en código fuente, realizado en un estudio previo

[MART94], como instrumento para recolectar los datos. Para que el analizador de código fuera aplicable a la investigación se necesitaron llevar a cabo modificaciones.

La investigación no experimental es aquella que se realiza sin manipular deliberadamente variables. Es decir, es investigación donde no hacemos variar intencionalmente las variables independientes. Lo que hacemos en la investigación no experimental es observar fenómenos tal y como se dan en su contexto natural, para después analizarlos. La investigación no experimental o expost-facto es cualquier investigación en la que resulta imposible manipular variables o asignar valores aleatoriamente a los sujetos o a las condiciones [HERN95].

5.3.2 Selección del Lenguaje de Programación Orientado a Objetos

El LPOO que se seleccionó para realizar el estudio fue el lenguaje C++ versión 3.1. La selección de este lenguaje se debe a que es uno de los LPOO más populares, además de que cumple con gran parte de las características del paradigma de la orientación a objetos.

5.3.3 Selección de la Muestra

Una de las consideraciones más importantes que nos conducen a obtener resultados más confiables al evaluar las métricas de Halstead, es eliminar la introducción de impurezas en los programas [HALS77]. Esto es, el programa debe estar escrito con una buena programación.

Para que la consideración anterior se cumpla de la mejor manera posible, se debe seleccionar la muestra de tal forma que garantice que los programas fueron escritos por programadores expertos. Por lo tanto, se seleccionaron como muestras los programas en código fuente que vienen como ejemplos en el paquete. En total se seleccionaron 611 programas.

5.3.4 Analizador de Código

El instrumento para obtener los datos empleados en la investigación fue un analizador de código fuente desarrollado en el estudio realizado por Martínez [MART94]. Este analizador fue construido de tal forma que es capaz de hacer análisis de código para cualquier lenguaje realizando pocas modificaciones. Para la aplicación de este analizador se realizaron algunas modificaciones, las cuales se pueden observar en el apéndice A.

Para la investigación, el analizador de código se alimentó con programas fuente escritos en C++ versión 3.1. La salida del analizador son las métricas de Halstead. Para validar el analizador se escogió una muestra pequeña de 10 programas pequeños, los cuales fueron alimentados al analizador. El resultado se comparó con el resultado de un proceso manual. No se encontró alguna diferencia entre ambos resultados. Así, el analizador está midiendo exactamente lo que queremos medir.

5.4 Resumen

En este capítulo se presentó el diseño de la investigación, se identificó el lenguaje de programación orientado a objetos que se va a analizar, así como la selección de la muestra y el instrumento de medición.

CAPÍTULO 6

ANÁLISIS DE LOS DATOS

6.1 Introducción

Este capítulo presenta el análisis de los datos que se llevó a cabo para contestar las hipótesis de la investigación. En 6.2 de muestran las estadísticas descriptivas. En 6.3 se analizan los datos para responder la primer hipótesis de la investigación. En 6.4 se analizan los datos para responder la segunda hipótesis de la investigación. En 6.5 se presenta un resumen del capítulo.

6.2 Estadísticas Descriptivas

La tabla VII nos muestra los valores reales (N) y los valores estimados ($\hat{N}$) de las longitudes de los programas.

Tabla VII

Longitudes Reales y Estimadas.

Prog.	N	$\hat{N}$
1	111	162,645
2	104	156,239
3	16	30,529
4	50	99,059
5	272	374,211
6	163	155,769
7	43	76,239
8	117	191,155

Prog.	N	$\hat{N}$
9	85	129,458
10	80	117,593
11	82	122,79
12	95	101,409
13	35	67,757
14	144	252,904
15	269	386,909
16	103	156,711

Prog.	N	$\hat{N}$
17	192	353,337
18	137	246,439
19	215	296,792
20	31	57,219
21	25	44,039
22	43	87,133
23	28	52,871
24	39	76,635

Tabla VII (Continuación)

Prog.	N	$\hat{N}$
25	16	19,61
26	101	155,925
27	40	86,522
28	22	35,61
29	78	96,657
30	25	44,039
31	67	76,635
32	16	19,61
33	22	32
34	70	77,303
35	25	40,139
36	67	72,106
37	16	19,61
38	22	32
39	69	139,742
40	13	12,755
41	28	53,564
42	30	57,705
43	27	48,729
44	18	24,406
45	15	17,51
46	13	12,755
47	13	13,61
48	201	263,303
49	58	67,02
50	64	76,635
51	13	12,755
52	28	53,564
53	30	57,705
54	17	23,219
55	26	43,651
56	41	76,107
57	15	17,51
58	46	107,541
59	64	125,458
60	13	12,755
61	13	13,61
62	58	135,258
63	133	233,12
64	117	241,524
65	88	120,768
66	88	120,768
67	48	83,651
68	629	522,162
69	473	427,058
70	45	113,93
71	48	82,603
72	45	114,968
73	51	89,138
74	47	78,255
75	51	88
76	100	204,093
77	49	91,823
78	126	227,917
79	59	123,164
80	122	221,592
81	59	123,73
82	80	187,909

Prog.	N	$\hat{N}$
83	43	57,705
84	50	125,458
85	95	216,694
86	50	123,164
87	43	113,112
88	51	112,506
89	44	72,106
90	80	133,487
91	49	81,073
92	93	155,769
93	49	81,073
94	46	76,239
95	47	76,107
96	46	76,239
97	22	35,61
98	78	96,657
99	25	44,039
100	67	76,635
101	16	19,61
102	137	156,239
103	22	35,61
104	13	12,755
105	28	53,564
106	30	57,705
107	12	9,51
108	21	31,02
109	15	17,51
110	16	20,265
111	13	13,61
112	29	39,51
113	28	44,039
114	1176	1229,023
115	34	82,603
116	64	154,15
117	601	503,514
118	134	228,639
119	43	57,705
120	24	31,261
121	34	62,671
122	47	98,016
123	221	265,963
124	178	259,412
125	50	123,164
126	43	113,112
127	51	112,506
128	51	125,458
129	46	71,549
130	70	97,219
131	20	27,651
132	115	186,985
133	26	48
134	113	195,312
135	107	188,987
136	130	188,987
137	29	57,219
138	16	16
139	32	40,139
140	47	71,273

Prog.	N	$\hat{N}$
141	70	101,623
142	33	52,871
143	41	62,671
144	86	135,258
145	46	88
146	34	62,054
147	154	174,7
148	222	325,568
149	41	81,325
150	292	326,732
151	22	40,139
152	17	27,119
153	85	127,706
154	40	82,603
155	25	49,663
156	13	12,755
157	52	91,823
158	62	64,913
159	77	117,593
160	109	141,127
161	27	68,813
162	79	143,258
163	77	149,316
164	20	31,261
165	18	27,651
166	18	27,651
167	41	72,106
168	71	118,078
169	58	86,159
170	44	81,325
171	52	88
172	24	48
173	30	57,059
174	54	98,016
175	47	72,106
176	27	58,529
177	29	57,219
178	34	68,813
179	50	71,273
180	17	23,219
181	25	43,651
182	31	48,729
183	103	155,769
184	48	76,239
185	31	48,729
186	59	99,059
187	64	145,042
188	31	48,729
189	67	150,842
190	95	102,054
191	60	91,357
192	44	76,107
193	55	91,357
194	135	191,698
195	67	91,357
196	49	76,239
197	76	145,542
198	21	39,51

Tabla VII (Continuación)

Prog.	N	Ñ
199	1284	1935,005
200	450	759,838
201	170	300,881
202	644	753,743
203	271	453,505
204	251	508,168
205	116	244,478
206	731	1043,922
207	176	262,988
208	553	732,252
209	377	609,375
210	161	307,907
211	169	250,702
212	401	658,454
213	230	425,79
214	664	947,814
215	102	168,642
216	156	227,32
217	336	579,04
218	320	564,414
219	185	307,207
220	377	702,218
221	590	863,423
222	340	550,507
223	590	800,054
224	695	1087,015
225	303	595,104
226	293	459,875
227	227	405,627
228	724	547,892
229	611	924,03
230	241	501,217
231	351	689,566
232	792	1131,547
233	467	503,001
234	389	551,304
235	106	145,542
236	114	145,542
237	143	197,869
238	115	178,818
239	85	139,742
240	115	156,239
241	112	139,742
242	320	490,289
243	77	167,297
244	776	1126,484
245	315	525,225
246	365	612,559
247	934	1155,871
248	1376	1485,214
249	662	929,696
250	898	935,016
251	1392	1578,102
252	195	339,697
253	204	374,211
254	1837	1429,399
255	261	455,167
256	528	761,251

Prog.	N	Ñ
257	772	1121,338
258	3002	2490,583
259	881	1023,61
260	340	460,941
261	118	232,713
262	239	263,303
263	557	739,084
264	1405	1166,16
265	210	340,303
266	775	836,956
267	740	679,526
268	218	294,847
269	2953	2548,228
270	2027	2299,987
271	1022	1043,079
272	144	223,067
273	117	197,869
274	238	400,713
275	111	202,149
276	247	276,096
277	1099	942,807
278	2972	1501,568
279	1566	1019,318
280	27	67,02
281	67	134,544
282	460	557,489
283	739	473,611
284	98	180,815
285	101	184,477
286	586	666,193
287	241	406,002
288	433	446,137
289	292	432,965
290	197	307,58
291	248	477,975
292	161	328,342
293	83	180,815
294	298	406,002
295	360	557,372
296	262	401,962
297	218	392,248
298	308	456,423
299	351	667,592
300	164	307,16
301	128	270,039
302	110	194,184
303	188	294,413
304	86	155,769
305	223	203,441
306	625	644,647
307	405	436,5
308	1063	783,108
309	23	35,61
310	53	96,22
311	93	123,164
312	73	131,327
313	87	116,239
314	1116	526,601

Prog.	N	Ñ
315	1178	533,938
316	445	419,008
317	739	587,291
318	672	564,501
319	439	405,552
320	741	579,04
321	612	543,258
322	455	398,842
323	811	608,142
324	682	571,487
325	839	580,405
326	1710	1455,819
327	2906	2960,902
328	1373	1262,177
329	850	637,974
330	2696	1639,777
331	1833	1499,377
332	2355	1361,798
333	1319	1186,161
334	624	607,596
335	4430	2793,124
336	1367	1548,771
337	2256	2535,811
338	442	491,677
339	4575	5093,766
340	2985	2806,6
341	105	112,106
342	168	140,344
343	577	501,217
344	298	373,684
345	476	543,258
346	717	950,16
347	52	72
348	37	48,181
349	3033	2560,784
350	466	461,974
351	143	302,789
352	70	145,542
353	427	847,766
354	256	190,346
355	1235	1499,515
356	2458	1603,612
357	646	611,807
358	98	134,014
359	346	269,212
360	90	167,149
361	886	902,973
362	619	622,197
363	54	111,89
364	18	44,039
365	18	44,039
366	93	92,529
367	18	44,039
368	18	44,039
369	18	44,039
370	18	44,039
371	18	44,039
372	18	44,039

Tabla VII (Continuación)

Prog.	N	$\hat{N}$
373	18	44,039
374	18	44,039
375	78	81,832
376	18	44,039
377	18	44,039
378	18	44,039
379	18	44,039
380	18	44,039
381	18	44,039
382	18	44,039
383	18	44,039
384	18	44,039
385	18	44,039
386	18	44,039
387	18	44,039
388	18	44,039
389	18	44,039
390	18	44,039
391	18	44,039
392	18	44,039
393	18	44,039
394	18	44,039
395	18	44,039
396	48	61,749
397	18	44,039
398	18	44,039
399	18	44,039
400	18	44,039
401	18	44,039
402	174	204,881
403	29	52,529
404	29	52,529
405	153	175,514
406	33	61,749
407	37	71,273
408	27	48,181
409	26	43,651
410	30	57,059
411	30	52,529
412	26	43,651
413	115	123,164
414	27	48
415	36	66,583
416	27	48
417	26	43,651
418	53	102,054
419	26	43,651
420	29	52,529
421	32	57,059
422	29	52,529
423	29	52,529
424	29	52,529
425	32	57,059
426	26	43,651
427	26	43,651
428	29	52,529
429	32	57,059
430	26	43,651
431	27	48
432	29	52,529
433	29	52,529
434	29	52,529
435	29	52,529
436	27	48
437	27	48
438	1196	1649,119
439	14	31,02
440	26	43,651
441	32	57,059
442	514	841,952
443	141	179,101
444	166	346,628
445	1120	1490,396
446	1309	1505,645
447	421	326,732
448	866	820,418
449	830	1232,882
450	76	168,042
451	909	1068,788
452	273	498,186
453	468	843,73
454	947	1194,844
455	2446	3598,867
456	2801	2681,445
457	415	669,265
458	757	918,142
459	733	775,588
460	1439	1754,958
461	292	392,248
462	1056	1529,039
463	1174	1383,058
464	902	1120,982
465	2386	2116,693
466	492	937,59
467	198	372,235
468	309	629,853
469	288	594,628
470	1262	1428,625
471	403	801,025
472	324	501,281
473	1571	1552,99
474	354	531,896
475	408	552,64
476	615	719,355
477	2308	1789,223
478	420	538,089
479	16	23,51
480	2875	2231,753
481	175	339,439
482	131	122,79
483	997	1425,768
484	97	167,149
485	181	238,308
486	727	799,636
487	940	1001,419
488	1237	1450,507
489	643	900,299
490	291	405,627
491	491	650,527
492	1259	1358,502
493	122	250,593
494	497	730,935
495	3018	3316,833
496	596	1089,976
497	700	1166,516
498	949	1444,855
499	368	629,853
500	61	113,93
501	139	281,763
502	294	533,938
503	469	855,562
504	592	1088,996
505	725	1294,094
506	773	1383,64
507	874	1541,512
508	1082	1720,79
509	1130	1789,223
510	1206	1903,214
511	1206	1911,204
512	1257	2070,429
513	1389	2266,077
514	1439	2347,535
515	1529	2452,76
516	389	522,623
517	523	723,929
518	325	460,215
519	475	614,668
520	1346	1050,483
521	914	1262,074
522	1182	1530,028
523	990	1388,54
524	428	585,885
525	492	709,212
526	619	1083,293
527	594	838,229
528	656	1206,658
529	347	564,414
530	253	551,689
531	461	708,851
532	429	793,306
533	2144	2398,46
534	486	871,3
535	1219	1351,304
536	1259	2038,898
537	749	1214,915
538	1212	2031,333
539	824	1676,18
540	490	938,115
541	588	720,536
542	1016	1745,606
543	932	1807,652
544	549	858,285
545	76	144,546
546	57	81,832
547	94	108,278

Tabla VII (Continuación)

Prog.	N	$\hat{N}$
548	41	48,729
549	596	850,58
550	573	813,05
551	145	269,263
552	1087	1550,342
553	56	117,303
554	1165	1281,873
555	1991	2919,347
556	422	530,424
557	963	1451,026
558	2904	3387,665
559	1292	1364,369
560	957	1403,086
561	1276	1965,299
562	738	1349,073
563	1058	1592,997
564	1237	1357,142
565	1037	1930,848
566	1250	1675,317
567	843	1415,063
568	1515	1675,447
569	1953	2005,624

Prog.	N	$\hat{N}$
570	1498	1881,463
571	1238	1265,799
572	509	1101,063
573	475	831,4
574	489	800,054
575	355	728,227
576	760	1186,708
577	2135	3121,381
578	1764	2311,124
579	3477	2853,53
580	1432	1838,503
581	913	1163,8
582	7367	4962,487
583	2250	2815,847
584	1197	1078,879
585	181	275,589
586	1489	1174,897
587	889	857,466
588	475	643,968
589	509	828,426
590	520	650,836
591	545	957,762

Prog.	N	$\hat{N}$
592	257	386,125
593	3110	3079,09
594	705	1083,636
595	3173	2864,204
596	668	1043,272
597	288	432,751
598	603	1175,65
599	101	168,642
600	1731	2204,2
601	216	392,402
602	258	501,408
603	331	529,144
604	469	798,407
605	831	1299,378
606	998	1521,05
607	1110	1723,898
608	1730	2195,856
609	2586	2862,169
610	771	938,115
611	2041	2321,06

La tabla VIII muestra los valores del nivel de lenguaje para los programas de muestra.

Tabla VIII

Niveles del Lenguaje.

Prog.	λ
1	0,652
2	0,827
3	1,215
4	0,947
5	0,623
6	1,643
7	3,152
8	0,746
9	0,84
10	1,171
11	1,225
12	1,711
13	2,166
14	0,721
15	1,286
16	0,691
17	0,699
18	1,121

Prog.	λ
19	0,495
20	4,085
21	3,544
22	1,68
23	2,769
24	1,973
25	5,194
26	1,111
27	2,215
28	3,533
29	1,082
30	2,713
31	1,083
32	5,194
33	2,191
34	0,771
35	1,836
36	0,843

Prog.	λ
37	5,194
38	2,191
39	1,101
40	5,132
41	2,78
42	3,166
43	2,43
44	2,746
45	5
46	5,132
47	5,839
48	1,149
49	1,114
50	1,131
51	5,132
52	2,78
53	3,166
54	6,275

Prog.	λ
55	5,307
56	3,25
57	5
58	1,89
59	1,467
60	5,132
61	5,839
62	1,322
63	0,778
64	0,577
65	0,655
66	0,655
67	1,154
68	0,392
69	0,298
70	1,236
71	1,26
72	1,048

Tabla VIII (Continuación)

Prog.	λ
73	1,101
74	1,268
75	1,193
76	0,963
77	2,809
78	0,709
79	1,882
80	0,731
81	1,614
82	0,891
83	1,719
84	1,329
85	0,753
86	1,595
87	1,592
88	1,709
89	1,494
90	2,167
91	2,299
92	1,763
93	2,299
94	2,226
95	2,901
96	2,226
97	3,533
98	1,082
99	2,713
100	1,083
101	5,194
102	0,663
103	3,533
104	5,132
105	2,78
106	3,166
107	7,755
108	4,705
109	5
110	3,17
111	5,839
112	2,484
113	1,945
114	0,514
115	2,16
116	1,001
117	0,735
118	0,703
119	1,719
120	2,744
121	3,072
122	2,047
123	0,372
124	1,094
125	1,595
126	1,592
127	1,709

Prog.	λ
128	1,169
129	1,573
130	0,823
131	3,615
132	0,703
133	5,136
134	0,479
135	0,686
136	0,487
137	3,096
138	7,68
139	10,965
140	5,098
141	1,363
142	1,836
143	1,646
144	0,838
145	1,465
146	2,122
147	0,669
148	0,42
149	2,597
150	0,638
151	4,136
152	6,534
153	1,362
154	1,43
155	2,296
156	5,132
157	1,908
158	14,339
159	2,689
160	4,435
161	2,381
162	0,485
163	0,439
164	4,065
165	5,083
166	5,083
167	1,894
168	0,85
169	1,44
170	1,63
171	0,931
172	6
173	3,475
174	0,985
175	1,222
176	1,759
177	3,822
178	1,333
179	2,241
180	6,275
181	3,693
182	1,476

Prog.	λ
183	1,952
184	2,057
185	1,476
186	11,603
187	1,182
188	1,476
189	1,046
190	0,889
191	1,56
192	3,488
193	1,747
194	0,915
195	1,334
196	2,1
197	0,951
198	4,997
199	0,431
200	1,379
201	0,89
202	1,28
203	1,226
204	0,827
205	1,286
206	0,995
207	0,896
208	0,621
209	0,971
210	1,851
211	0,857
212	0,738
213	0,98
214	2,253
215	0,615
216	0,531
217	0,882
218	0,852
219	0,837
220	1,029
221	1,158
222	0,76
223	0,559
224	0,716
225	1,351
226	0,728
227	0,899
228	0,806
229	1,04
230	0,793
231	1,127
232	0,524
233	0,254
234	0,452
235	2,416
236	2,284
237	2,425

Prog.	λ
238	1,562
239	2,882
240	2,132
241	2,106
242	1,168
243	2,035
244	0,627
245	1,68
246	1,418
247	0,578
248	0,465
249	0,693
250	0,385
251	0,333
252	1,076
253	1,569
254	0,425
255	1,309
256	0,659
257	0,399
258	0,202
259	0,505
260	4,479
261	1,953
262	0,585
263	1,226
264	0,417
265	1,13
266	0,456
267	0,296
268	0,477
269	0,35
270	0,302
271	0,222
272	0,478
273	1,125
274	0,734
275	2,447
276	0,465
277	0,339
278	0,15
279	0,199
280	2,561
281	1,08
282	0,449
283	0,342
284	0,959
285	2,267
286	0,274
287	1,726
288	0,464
289	0,968
290	0,668
291	3,543
292	0,815

Tabla VIII (Continuación)

Prog.	λ
293	0,947
294	0,509
295	0,891
296	8,835
297	0,944
298	1,572
299	3,4
300	2,018
301	1,633
302	0,543
303	1,276
304	1,114
305	0,444
306	0,318
307	0,947
308	0,189
309	3,694
310	1,212
311	0,939
312	0,836
313	1,09
314	1,009
315	0,9
316	0,893
317	0,791
318	0,557
319	0,833
320	0,638
321	0,531
322	0,79
323	0,628
324	0,492
325	0,385
326	0,462
327	0,353
328	0,543
329	0,718
330	0,355
331	1,398
332	0,333
333	0,484
334	0,501
335	0,23
336	0,514
337	0,405
338	1,241
339	0,31
340	0,253
341	3,054
342	1,07
343	0,503
344	0,633
345	0,371
346	0,42
347	0,878

Prog.	λ
348	2,487
349	0,386
350	1,453
351	2,87
352	3,363
353	8,227
354	0,573
355	0,534
356	0,194
357	0,165
358	0,828
359	0,483
360	2,194
361	0,715
362	0,303
363	0,683
364	3,473
365	3,473
366	5,681
367	3,473
368	3,473
369	3,473
370	3,473
371	3,473
372	3,473
373	3,473
374	3,473
375	5,052
376	3,473
377	3,473
378	3,473
379	3,473
380	3,473
381	3,473
382	3,473
383	3,473
384	3,473
385	3,473
386	3,473
387	3,473
388	3,473
389	3,473
390	3,473
391	3,473
392	3,473
393	3,473
394	3,473
395	3,473
396	3,933
397	3,473
398	3,473
399	3,473
400	3,473
401	3,473
402	0,303

Prog.	λ
403	3,746
404	3,746
405	3,999
406	3,754
407	4,654
408	3,226
409	3,841
410	4,135
411	5,13
412	3,841
413	2,731
414	4,32
415	5,501
416	4,32
417	3,841
418	1,471
419	3,841
420	3,746
421	4,411
422	3,746
423	3,746
424	3,746
425	4,411
426	3,841
427	3,841
428	3,746
429	4,411
430	3,841
431	4,32
432	3,746
433	3,746
434	3,746
435	3,746
436	4,32
437	4,32
438	0,764
439	5,577
440	3,841
441	4,411
442	1,199
443	2,548
444	1,535
445	0,53
446	0,452
447	0,526
448	0,272
449	0,856
450	0,992
451	0,451
452	2,087
453	0,775
454	0,44
455	0,719
456	0,448
457	1,384

Prog.	λ
458	0,784
459	0,228
460	0,628
461	0,702
462	0,481
463	0,535
464	0,633
465	0,244
466	0,947
467	0,943
468	1,117
469	1,186
470	0,339
471	1,067
472	0,715
473	0,426
474	1,083
475	0,887
476	0,612
477	0,339
478	1,003
479	2,625
480	0,233
481	1,1
482	0,704
483	0,619
484	1,574
485	1,084
486	0,611
487	0,887
488	0,484
489	0,772
490	0,645
491	0,492
492	0,376
493	1,884
494	0,422
495	0,389
496	8,81
497	3,537
498	0,739
499	1,379
500	1,283
501	1,846
502	2,625
503	1,451
504	1,28
505	0,955
506	0,877
507	0,846
508	0,763
509	0,748
510	0,81
511	0,793
512	0,877

Tabla VIII (Continuación)

Prog.	λ
513	0,903
514	0,926
515	0,847
516	1,379
517	1,005
518	0,624
519	0,541
520	0,24
521	0,616
522	0,482
523	0,713
524	0,72
525	0,6
526	0,64
527	0,971
528	1,309
529	0,841
530	1,437
531	0,635
532	1,307
533	0,662
534	1,163
535	0,47
536	0,82
537	0,889
538	4,973

Prog.	λ
539	2,907
540	1,517
541	2,825
542	4,295
543	1,242
544	0,621
545	2,629
546	2,175
547	0,969
548	3,69
549	0,4
550	0,444
551	1,001
552	0,281
553	2,714
554	0,295
555	0,448
556	0,724
557	0,298
558	0,485
559	0,384
560	0,359
561	0,526
562	1,823
563	0,974
564	0,632

Prog.	λ
565	0,796
566	0,722
567	2,639
568	1,581
569	1,473
570	1,761
571	1,797
572	2,071
573	1,459
574	0,766
575	1,684
576	0,57
577	2,099
578	0,491
579	0,212
580	0,357
581	0,923
582	0,11
583	0,302
584	1,572
585	0,835
586	0,489
587	1,013
588	0,593
589	0,839
590	0,457

Prog.	λ
591	0,654
592	0,511
593	0,362
594	0,533
595	0,336
596	0,512
597	0,954
598	5,109
599	0,644
600	0,376
601	1,155
602	0,947
603	0,61
604	0,612
605	0,554
606	0,511
607	0,568
608	0,382
609	0,379
610	0,641
611	0,566

La tabla IX muestra la media y desviación estándar para el nivel del lenguaje del C++.

Tabla IX

Media y Desviación Estándar.

Lenguaje	Media	Desviación Estándar
C++	1.84348	1.717233

La tabla X muestra la tabla de frecuencias para el nivel del lenguaje del C++.

Tabla X

Frecuencias para el Nivel del Lenguaje.

Intervalo	Frecuencia	Porcentaje
[0-1]	271	44,3535%
[1-2]	140	22,9133%
[2-3]	59	9,6563%
[3-4]	82	13,4206%
[4-5]	20	3,2733%
[5-6]	27	4,4190%
[6-7]	4	0,6547%
[7-8]	2	0,3273%
[8-9]	3	0,4910%
[9-10]	0	0,0000%
[10-11]	1	0,1637%
[11-12]	1	0,1637%
[12-13]	0	0,0000%
[13-14]	0	0,0000%
[14-15]	1	0,1637%
TOTAL	611	100,0000%

6.3 Análisis de Datos de la Primera Hipótesis de Investigación

La primer pregunta de investigación es: ¿Es el estimador $\hat{N}$ un buen estimador de N, en lenguajes orientados a objetos? Para responder a esta pregunta se calcula el coeficiente de correlación de Pearson entre N y $\hat{N}$ (de la misma forma que lo realizó Halstead).

El coeficiente de correlación de Pearson (r) mide la fuerza de la relación lineal que existe en una muestra de n datos bivariados. Algunas de sus características son [KVAN89]:

1) Los valores de r están en rango [-1,1].

2) Entre más grande sea |r| (valor absoluto de r), más fuerte es la relación lineal.

3) Si el valor de r es cercano a cero, indica que no existe una relación lineal entre las variables.

4) Si r = 1 o r = -1 implica que existe un patrón lineal perfecto entre las variables de la muestra.

5) Los valores de r = 0, r = 1 o r = -1 son muy raros en la práctica.

En la tabla XI se muestra el resultado del coeficiente de correlación de Pearson para C++.

Tabla XI

Coeficiente de Correlación de Pearson entre N y $\hat{N}$.

Lenguaje	r
C++	0.93374

El resultado del análisis de correlación indica que N y $\hat{N}$ están fuertemente correlacionados. Por lo tanto, se puede concluir que para el lenguaje orientado a objetos, C++, $\hat{N}$ es un buen estimador de N.

6.4 Análisis de Datos de la Segunda Hipótesis de Investigación

La segunda hipótesis de investigación es: ¿Es el nivel del lenguaje de los LPOO's mayor que el nivel del lenguaje de los 3GL's de propósito general y menor que el nivel del lenguaje de los 4GL's?

La investigación hipotetiza que el nivel del lenguaje del LPOO es mayor que 1.53 pero menor que 1.9544 y 1.9763. Esto valores se tomaron así porque el valor mayor nivel del lenguaje para un lenguaje de tercera generación es de 1.53 (PL/1) [HALS77], y los valores de 1.9544 y 1.9763 son los valores para los lenguajes de

cuarta generación DBaseIII y Foxpro2 respectivamente obtenidos en el estudio realizado por Martínez [MART94].

Para demostrar la hipótesis de investigación se realizaron pruebas Z para analizar las medias de los niveles del lenguaje. La prueba Z se utiliza para probar hipótesis sobre la media de una población cuando se tiene una muestra grande y para probar hipótesis sobre la media de muestras independientes [KVAN89].

La tabla XII muestra el resultado de la prueba Z entre la siguiente pareja de hipótesis:

H_o : la media del nivel del lenguaje para los LPOO's es menor o igual a 1.53.

H_a : la media del nivel del lenguaje para los LPOO's es mayor a 1.53.

Tabla XII

Resultado de la Prueba Z entre el LPOO y los 3GL's.

Lenguaje	Z	Nivel de significancia
C++	4.5123	0.000003209

El resultado de la tabla XII indica que para el LPOO C++ existe suficiente evidencia que apoye la H_a, es decir, existe suficiente evidencia para concluir que el nivel del lenguaje para el lenguaje C++ es mayor que 1.53.

La tabla XIII muestra el resultado de la prueba Z entre la siguiente pareja de hipótesis:

H_o : la media del nivel del lenguaje para los LPOO's es mayor o igual a 1.9544.

H_a : la media del nivel del lenguaje para los LPOO's es menor a 1.9544.

Tabla XIII

Resultado de la Prueba Z entre el LPOO y el Nivel del Lenguaje para DBaseIII.

Lenguaje	Z	Nivel de significancia
C++	-1.5966	0.055177448

El resultado de la tabla XIII indica que para el LPOO C++ no existe suficiente evidencia que apoye la H_a, es decir, no existe suficiente evidencia para concluir que el nivel del lenguaje para el lenguaje C++ es menor que 1.9544.

La tabla XIV muestra el resultado de la prueba Z entre la siguiente pareja de hipótesis:

H_o : la media del nivel del lenguaje para los LPOO's es mayor o igual a 1.9763.

H_a : la media del nivel del lenguaje para los LPOO's es menor a 1.9763.

Tabla XIV

Resultado de la Prueba Z entre el LPOO y el Nivel del Lenguaje para Foxpro2.

Lenguaje	Z	Nivel de significancia
C++	-1.9119	0.027944443

El resultado de la tabla XIV indica que para el LPOO C++ existe suficiente evidencia que apoye la H_a, es decir, existe suficiente evidencia para concluir que el nivel del lenguaje para el lenguaje C++ es menor que 1.9763.

6.5 Resumen

En este capítulo se presentó el análisis de los datos que se recolectaron. Los resultados que se obtuvieron son:

1) Existe una fuerte correlación entre $\hat{N}$ y N, para lenguajes orientados a objetos, lo cual indica que $\hat{N}$ es un buen estimador de N.

2) Existe suficiente evidencia estadística que indica que el nivel del lenguaje del C++, es mayor que el nivel del lenguaje de los 3GL's.

3) No existe suficiente evidencia estadística que indique que el nivel del lenguaje del C++ sea menor que el nivel del lenguaje para el DBaseIII.

4) Existe suficiente evidencia estadística que indica que el nivel del lenguaje del C++ es menor que nivel del lenguaje del Foxpro2.

CAPÍTULO 7

CONCLUSIONES

En este capítulo se presentan los resultados del análisis de datos que se presentaron en el capítulo anterior. Además se dan las conclusiones y algunas sugerencias para investigaciones futuras.

7.1 Objetivos de la Investigación

Los principales objetivos del estudio fueron:

1) Determinar si el estimador de la longitud de un programa propuesto por Halstead es un buen estimador para lenguajes de programación orientados a objetos.

2) Determinar si el nivel del lenguaje de los lenguajes de programación orientados a objetos es mayor que el nivel del lenguaje de los 3GL's, pero menor que el nivel del lenguaje de los 4GL's

7.2 Discusión, Conclusiones y Sugerencias de Investigaciones Futuras del Objetivo 1

Se encontró que el estimador de la longitud propuesto por Halstead, es un buen estimador de la longitud de un programa para el lenguaje de programación orientado a objetos C++. Esto es válido en el rango de las longitudes del programa de 12 a 7367.

El análisis de correlación lineal nos indica que existe una fuerte correlación lineal entre N y $\hat{N}$, en la tabla VII puede observarse que $\hat{N}$ empieza sobreestimando la longitud del lenguaje y termina subestimandola.

La relación entre N y $\hat{N}$ tiene una representación gráfica como se muestra en la figura 13. Esta relación tiene el mismo comportamiento que se observó en los estudios de Shen [SHEN83] y Martínez [MART94].

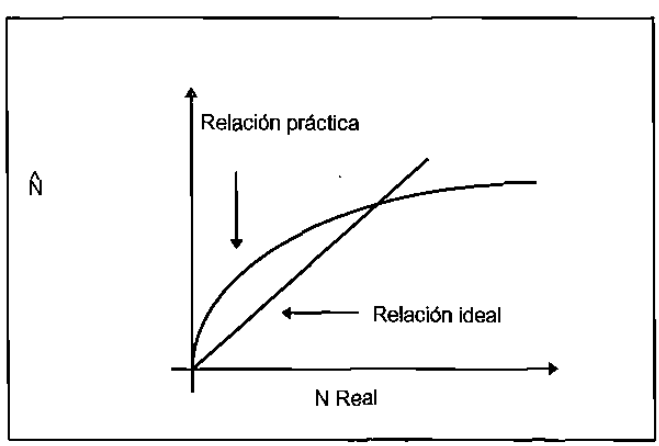


Figura 13 Relación Ideal entre N y $\hat{N}$, y Relación Práctica.

Una de las sugerencias para las investigaciones futuras sería tratar de generalizar más este resultado, es decir aplicarlo a otros lenguajes de programación orientados a objetos, tales como Object Pascal o Smalltalk.

7.3 Discusión, Conclusiones y Sugerencias de Investigaciones Futuras del Objetivo 2

Se encontró que el nivel del lenguaje del C++ es mayor que el nivel del lenguaje de los 3GL's y menor que el nivel del lenguaje del Foxpro2, que es un lenguaje de cuarta generación. Con respecto al lenguaje DBaseIII no se encontró la suficiente evidencia estadística que indique que el nivel del lenguaje del C++ sea menor que el nivel del lenguaje del DBaseIII, pero numéricamente se encontró que sí es menor con una diferencia de 5.67%. Esto quiere decir que las capacidades de orientación a objetos del C++ facilita la programación en lenguajes de tercera generación con características de orientación a objetos. Esta facilidad es gracias a los conceptos que se trataron en el capítulo 4, siendo uno de los más importantes el de la herencia.

Aunque la clasificación del nivel del lenguaje del C++ es la clasificación esperada, podemos observar que el nivel del lenguaje para el LPOO analizado no es constante. En el estudio realizado por Shen [SHEN83] y en el de Martínez [MART94] se observó que el nivel del lenguaje depende de la longitud del programa. Una representación gráfica se muestra en la figura 14. En esta investigación el nivel del lenguaje del C++ se comportó de manera similar que en las investigaciones anteriores.

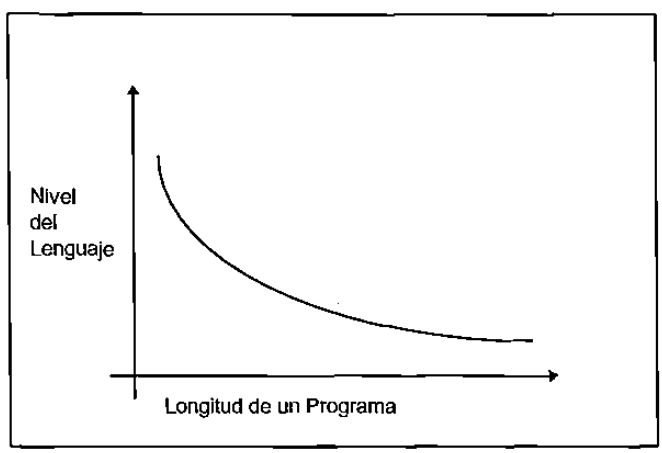


Figura 14 Comportamiento del Nivel del Lenguaje.

Una de las sugerencias para las investigaciones futuras sería aplicar estas métricas a otros lenguajes de programación orientados a objetos, tales como, Object Pascal, Smalltalk, etc., con el objetivo de generalizar el nivel del lenguaje para los LPOO's.

Otra sugerencia sería aplicar estas métricas a los programas que se desarrollan en las casas de software, con el objetivo de obtener el nivel del lenguaje de tales organizaciones.

Otra posible investigación sería la aplicación de estas métricas con los alumnos de escuelas de programación, con el objetivo de obtener una medida para poder comparar el nivel del lenguaje de cada una de las escuelas y en base a esto decir qué escuela tiene un mejor método de enseñanza.

REFERENCIAS

[ATHE88] Athey, Thomas H. y Zmud Robert W. Introduction to Computers and Information Systems, Second Edition. Scott, Forest and Company. (1988).

[BOOC86] Booch, Grady. "Object-Oriented Development". IEEE Transactions on Software Engineering, Vol. SE-12, No. 2. (February 1986).

[BOWM90] Bowman, Brent J. y Newman William A. "Software Metrics as a Programming Training Tool". J. Systems Software. (1990).

[BROO94] Brooks, Christopher L. y Buell, Christopher G. "A Tool for Automatically Gathering Object-Oriented Metrics". IEEE (1994).

[BURC92] Burch, Jonh G. Systems Analysis, Desing, and Implementation. Boyd & Fraser Publishing Company. (1992).

[CHID94] Chidamber, Shyam R. y Kemerer, Chris F. "A Metrics Suite for Object-Oriented Design". IEEE Transactions on Software Engineering, Vol. 20, No. 6. (June 1994).

[FENT94] Fenton, Norman. Software Measurement: "A Necesary Scientific Basis". IEEE Transactions on Software Engineering, Vol. 20, No. 3. (March 1994).

[FICH92] Fichman, Robert G. y Kemerer Chris F. "Object-Oriented and Conventional Analysis and Design Methodologies: Comparison and Critique". IEEE Computer. (October 1992).

[FREN92] Frenzel, Carroll W. Management of Information Technology. Boyd & Fraser Publishing Company. (1992).

[HALS77] Halstead, H. Maurice. Elements of Software Science. North Holland New York. (1977)

[HERN95] Hernández Samperi, Roberto, Fernández Collado, Carlos y Baptista Lucio, Pilar. Metodología de la Investigación. Mc Graw Hill. (1995).

[JAME87] Senn, James A. Information Systems in Management, Third Edition. Wadsworth Publishing Company. (1987).

[JONE91] Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality. Mc Graw Hill. (1991).

[JONH90] Jonhson, G. Vaughn. Information Systems: A Strategic Approach. Mountain Top Publishing. (1990).

[KHAN95] Khan, Emdad H., Al-A'ali, Mansoor y Girgis, Moheb R. "Object-Oriented Programming for Structured Procedural Programmers". IEEE Computer. (October 1995).

[KVAN89] Kvanli, Alan H., Guynes, C. Stephen and Pavur, Robert J. Introduction to Business Statistics: A Computer Integrated Approach, Second Edition. West Publishing Company. (1989).

[LARA90] Laranjeira, Luiz A. "Software Size Estimation of Object-Oriented Systems". IEEE Transactions on Software Engineering, Vol. 16, No. 5. (May 1990).

[LOKA96] Lokan, Christopher J. "Early Size Prediction for C and Pascal Programs". J. Systems Software. (1996).

[MACI94] Macías Guerrero, Juan Antonio. "Estudio Comparativo de Lenguajes de Programación Orientados a Objetos Basado en las Facilidades para Implantar el Concepto de Objeto". Tesis de Maestría en Ciencias. Instituto Tecnológico y de Estudios Superiores de Monterrey. Monterrey, N.L. México. (Mayo de 1994)

[MART94] Martínez Flores, José Luis. "Métricas de Software en Lenguajes de Cuarta Generación". Tesis de Maestría en Ciencias de la Administración Especialidad en Sistemas. UANL FIME San Nicolás de los Garza, N.L. México. (Marzo de 1994).

[MCCA76] McCabe, Thomas J. "A Complexity Measure". IEEE Transactions on Software Engineering, Vol. SE-2, No. 4. (December 1976).

[MÖLL93] Möller, K. H. y Paulish, D.J. Software Metrics: A Practitioner's Guide to Improved Product Development. Chapman & Hall Computing. (1993).

[PRES93] Pressman, Roger S. Ingeniería del Software: Un Enfoque Práctico, Tercera Edición. Mc Graw Hill. (1993).

[RINE92] Rine, David C. y Bhargava, Bharat. "Object-Oriented Computing". IEEE Computer. (October 1992).

[SHEN83] Shen Vincent Y., Conte, Samuel D. y Dunsmore, H.E. "Software Science Revisted: A Critical Analisys of the Theory and Its Empirical Support". IEEE Transactions on Software Engineering, Vol. SE-9, No. 2. (March 1983).

[TEGA92] Tegarden, David P. y Sheetz, Steven D. "Effectiveness of Traditional Software Metrics for Object-Oriented Systems". IEEE. (1992).

[VERN92] Verner, June y Tate, Graham. "A Software Size Model". IEEE Transactions on Software Engineering, Vol. 18, No. 4. (April 1992).

[WEGN92] Wegner, Peter. "Object-Oriented Modeling". IEEE Computer (October 1992).

[WINB93] Winbald, Ann L., Edwards, Samuel D. y King, David R. Software Orientado a Objetos. Addison-Wesley/Díaz Santos (1993).

[WRIG91] Wrigley, Clive D. y Dexter, Albert S. "A Model for Measuring Information System Size". MIS Quarterly. (June 1991).

[YOUR94] Yourdon, Edward. Object-Oriented Systems Design. Yourdon Press, Prentice Hall Building. (1994).

APÉNDICE A

Modificación del Analizador de Código

APÉNDICE A

MODIFICACIÓN DEL ANALIZADOR DE CÓDIGO

El analizador de código que se utilizó en el estudio de Martínez [MART94] fue desarrollado para analizar el código fuente de los lenguajes Foxpro2 y DBaseIII, pero dicho analizador se puede modificar para poder analizar el código fuente de otros lenguajes.

Las modificaciones que se realizaron fueron:

1) Cambiar el archivo de operadores, para que contenga los operadores del lenguaje C++.

2) Cambiar el archivo de las palabras que no tienen efecto en la ejecución del programa para el lenguaje C++.

3) Cambiar en la codificación del analizador la forma en que se llevan a cabo los comentarios en el lenguaje C++.

4) Agregar un módulo que identifique las funciones definidas por el usuario y las llamadas a los objetos, las cuales se consideran como operadores, ya que estas funciones realizan una acción. En la figura 15 se muestra más claramente esta modificación.

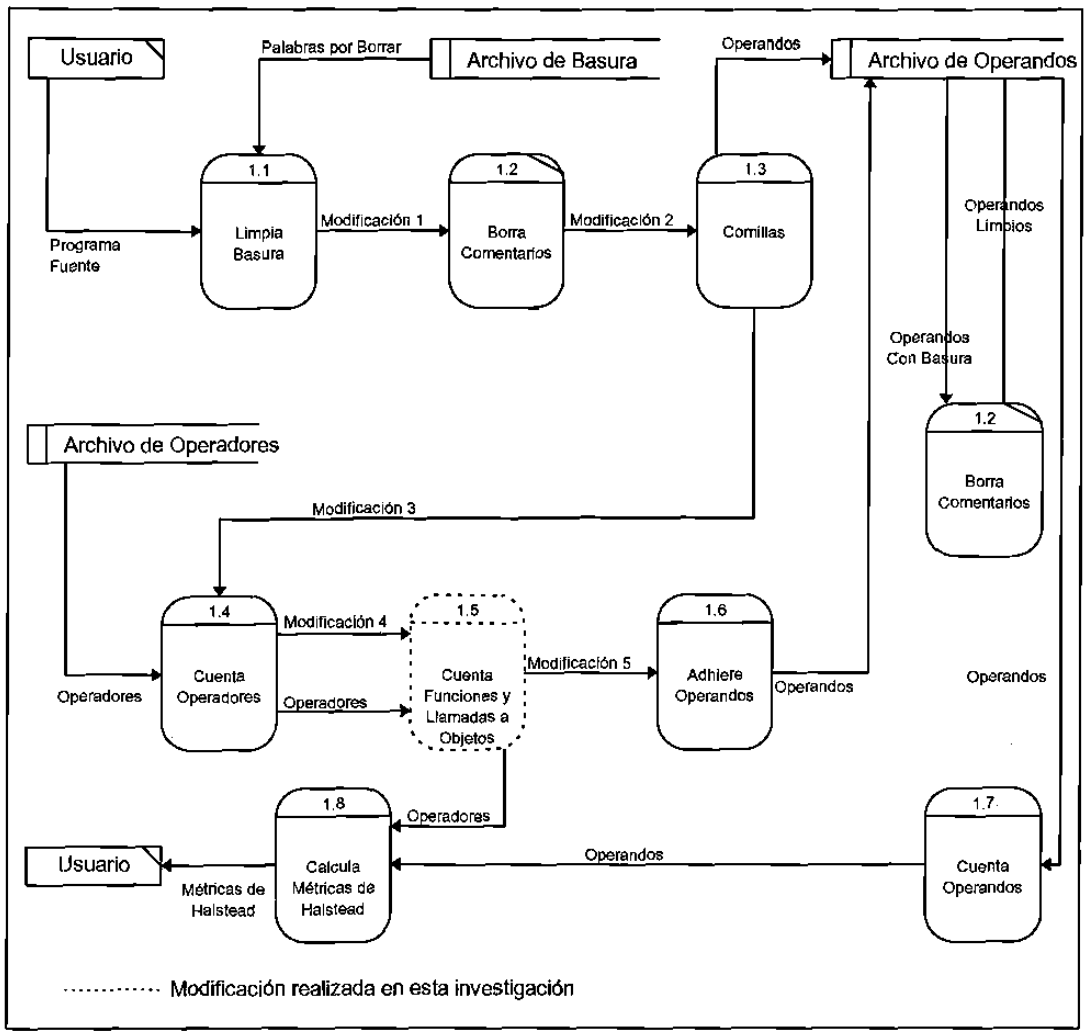


Figura 15 Diagrama de Flujo de Datos de Nivel 1 del Analizador de Código.

A continuación ofrecemos el significado de algunos de los nombres que intervienen en el diagrama de flujo de datos:

1) Archivo Basura.- Es el archivo que contiene las palabras que no tienen efecto en el conteo.

2) Limpia Basura.- es el proceso que elimina la basura de un programa.

3) Modificación 1.- Programa sin basura.

4) Borra Comentarios.- Es el proceso que elimina los comentarios, espacios en blanco y tabulaciones, dentro de un programa.

5) Modificación 2.- Programa sin comentarios, espacios en blanco y tabulaciones.

6) Comillas.- Es el proceso que elimina los operandos que vienen entre comillas dentro de un programa.

7) Modificación 3.- Programa sin comillas.

8) Modificación 4.- Programa sin operadores.

9) Modificación 5.- Programa sin funciones definidas por el usuario y sin llamadas a objetos.

RESUMEN AUTOBIOGRÁFICO

Xavier Espinosa de los Monteros Anzaldúa

Candidato para el Grado de

Maestro en Ciencias de la Administración con Especialidad en Sistemas

Tesis: MÉTRICAS DE HALSTEAD APLICADAS A LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

Campo de Estudio: Métricas de Software.

Biografía:

Nacido en Monterrey, N.L., el 12 de Mayo de 1973; hijo de Enrique Espinosa de los Monteros Martínez y María Elma Anzaldúa Hernández.

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 1994.