Un Ambiente para evaluación de Sistemas de Software mediante métricas clásicas

Lic. P. Pesado[1] , A.C. C. Borroni[2] , A.C. J. Zorzano[2], Ing. A.De Giusti[3]

Laboratorio de Investigación y Desarrollo en Informática[4]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Se presenta un ambiente para evaluación de sistemas de software mediante recolección de métricas y estimaciones.

El objetivo principal es conformar una línea base a partir de mediciones efectuadas sobre desarrollos ya implementados que sirva de información para futuras estimaciones.

Sin duda las mejores estimaciones surgen del análisis de proyectos similares efectuados en la organización. Las experiencias recogidas de desarrollos análogos realizados por otras organizaciones pueden servir de indicativos pero hay factores puntuales (recursos humanos, tecnológicos, complejidad del problema etc.) que son difíciles de generalizar y distorsionan las posibles estimaciones.
Es por eso la importancia de que los planificadores de proyectos cuenten con información real recolectada de fuentes propias.

Un segundo objetivo está ligado a los procesos de mantenimiento en los que es útil conocer la complejidad lógica, tamaño y flujo de información a fin de identificar los posibles módulos críticos en un desarrollo.

Por último tenemos un tercer objetivo que se relaciona con los procesos de reingeniería que requieren conocimiento del código fuente como base para el reanálisis del problema.

El ambiente presentado combina la recolección de métricas automáticas a partir de código fuente ya desarrollado, con información ingresada por el ingeniero de sistemas, permitiendo especificar nuevas mediciones orientadas al problema.

[1] Prof. Titular. Ded. Excl. LIDI. Dpto. de Informática, Facultad de Cs. Exactas, UNLP. Profesional CIC.
E-mail ppesado@ada.info.unlp.edu.ar
[2]Analista de Computación. Alumno de Lic. en Informática, Facultad de Ciencias Exactas, UNLP.
E-mail borroni@ada.info.unlp.edu.ar
[3] Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.
E-mail degiusti@ada.info.unlp.edu.ar
[4] Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono 54-21-227707
E-mail lidi@ada.info.unlp.edu.ar

Introducción

La gestión de un proyecto de software abarca todo el proceso de desarrollo de un sistema de principio a fin, es decir desde el momento mismo de definición del problema hasta el momento en que se abandona el software.

Una mala gestión de proyecto se manifiesta principalmente en incumplimiento de plazos, incremento de los costos y/o por la entrega de productos de software de mala calidad (lo que redunda en insatisfacción del usuario y gran cantidad de tiempo dedicado al mantenimiento). Estos resultados tienen en general una consecuencia negativa que afecta a cualquier organización: el perjuicio económico.

Para poder mejorar el proceso de desarrollo del software y el software en sí mismo (por ejemplo en términos de calidad o productividad) debemos reconocer los problemas que lo afectan. La obtención de métricas que reflejen el estado actual del proceso de desarrollo de software da al gestor de proyecto una base cuantitativa para plantear objetivos que hagan al mejoramiento de dicho proceso. [ART93]

Más aún, las métricas establecidas en una línea base, permitirán contrastar los nuevos resultados con los anteriores y así poder evaluar la evolución de la organización de modo objetivo.

De Marco [DEM82] enuncia que "No se puede controlar lo que no se puede medir". Cualquier aspecto del proyecto que se desee controlar debe ser medido. De no ser así, se carece de la retroalimentación necesaria para poder controlar.

Una de las actividades cruciales de la gestión de proyecto es la planificación y para poder planificar un proyecto deben obtenerse estimaciones del esfuerzo humano requerido, de la duración cronológica del proyecto y del costo del proyecto. [BOE81]

Dentro de este contexto, la detección temprana de discordancias entre lo real y lo estimado, es de vital importancia para mantener un proyecto bajo control.

Al principio, el costo del software constituía un pequeño porcentaje del costo total de los sistemas informáticos basados en computadora. Un error considerable en las estimaciones del costo del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas informáticos. Un gran error en la estimación del costo puede ser lo que marque la diferencia entre beneficios y pérdidas. Excederse en el costo puede ser desastroso para el equipo de desarrollo. [PRE93].

La estimación del costo y el esfuerzo del software no es una ciencia exacta. Sin embargo, valiéndonos de un proceso sistemático apoyado en datos históricos confiables se pueden obtener estimaciones con un grado de riesgo aceptable.

La calidad del software es generalmente un objetivo siempre presente en un proyecto software.

Recolectar métricas de aspectos de calidad de software puede servir para detectar que factores afectan de manera significativa la calidad del software y permitir a una organización adecuar su proceso de Ingeniería de software a fin de eliminar las causas de los defectos que tienen el mayor impacto en el desarrollo de software. [HUM89]

También pueden usarse para detectar diseños o implementaciones insatisfactorias en términos de calidad y obligar a una revisión de los mismos.

Al finalizarse la codificación del sistema, las métricas pueden dar una base para saber por ejemplo cuáles módulos serán más propensos a error o cuántos errores se pueden esperar al inicio de las pruebas del software, información que resulta sumamente útil para la planificación de las pruebas.

La ingeniería inversa y la reingeniería que actualmente están siendo puestas en práctica para reciclar viejos sistemas, utilizan la recolección de métricas para detectar partes de dicho sistema que requerirán especial esfuerzo para ser mejorados en términos de calidad.

Durante el mismo proceso de reingeniería, las mediciones nos darán una indicación de si efectivamente se está obteniendo un producto de mejor calidad, lo que nos servirá para evaluar dicho proceso.

Las métricas de software apuntan a un amplio rango de medidas sobre el software de computadoras.

Existen *métricas del proceso* y *métricas del producto*.

- Las métricas del proceso son aquellas que cuantifican atributos del proceso de desarrollo de software y del equipo de desarrollo.
- Las métricas del producto se aplican al producto software resultante del proceso de desarrollo (ej: tamaño, complejidad).

Dentro del contexto de la planificación del proyecto de software se distinguen principalmente las *métricas de productividad* y *métricas de calidad*.

- Las métricas orientadas a la productividad se centran en el rendimiento del proceso de Ingeniería de Software.
- Dentro de las métricas orientadas a la calidad existen las que dan una indicación de la concordancia entre el software concebido y los requisitos explícitos e implícitos del cliente (ej: fiabilidad) y otras que dan parámetros técnicos del software (ej: complejidad ciclomática [MCC76]).

Las mediciones pueden englobarse en dos categorías: *medidas directas* e *indirectas*.

- Las medidas directas son aquellas que se obtienen del proceso o producto observándolo y que no dependen de otras medidas. Entre las medidas directas del proceso de Ingeniería de Software se encuentran el costo y el esfuerzo. Entre las medidas directas del producto software se encuentran las líneas de código, el tiempo de ejecución, el tamaño de memoria requerido y los defectos observados en un determinado período de tiempo.
- Las medidas indirectas por el contrario dependen a su vez de medidas más elementales. Entre las medidas indirectas del producto se encuentran la funcionalidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento.

Desde otro punto de vista existen *métricas objetivas* y *subjetivas*.

- Una medida subjetiva es aquella cuyo valor depende de una apreciación personal de quien la aplica.
- Una medida objetiva es aquella que puede ser computada en forma precisa por un algoritmo (su valor no cambia con el tiempo, lugar u observador).

Obviamente, el proceso de medición y evaluación no es un trabajo sencillo, es una tarea dura que requiere de un esfuerzo constante y que brinda frutos después de mucho tiempo.

Grady y Caswell en su libro [GRA87] describen trabajos de varios años para disponer de tendencias organizativas en este tema.

La obtención de datos, el cálculo de métricas y la evaluación de datos son los tres pasos que hay que implementar para establecer una línea base de mediciones.

La línea base consiste en datos recogidos de anteriores proyectos de desarrollo de software.

La recolección de datos objetivos puede efectuarse en forma manual o automática. Las estrategias básicas para llevar a cabo la recolección se basan en analizadores de software, formularios de reportes, entrevistas y ambientes que recolectan datos en el momento de la codificación.

El costo de recolectar las métricas obviamente no debería superar los beneficios que resulten de su recolección. Las métricas más económicas en general son las recolectadas vía analizadores de código que requieren poca intervención de recursos humanos y no obstruyen ni demoran los procesos de desarrollo.

Objetivo

Este desarrollo se ha realizado en el marco de un trabajo de Tesina de la Licenciatura en Informática y el producto final ha sido un ambiente que permite la recolección de métricas sobre desarrollos de software.

La utilidad de contar con el ambiente se centra en la posibildad de recolectar datos propios extraídos de proyectos realizados, poder comparar y combinar métricas y dar la libertad al usuario de definir sus propias métricas a partir de mediciones elementales.

La conformación de un conjunto de mediciones en proyectos realizados permitirá la utilización de los resultados para estimaciones de desarrollos futuros.

Métricas analizadas

La tarea de medición puede realizarse en distintas etapas del proceso de desarrollo del software, es por ello que puede ser conveniente clasificar a las métricas asociándolas con la actividad del ciclo de desarrollo en que pueden aplicarse.

Para la *etapa de especificación de requerimientos* De Marco sugiere utilizar la "function bang", calculando que el esfuerzo necesario para realizar un proyecto es función de parámetros extraídos de los modelos funcional (funciones no explotadas, datos primitivos del diccionario de datos), de datos (entidades y relaciones), y dinámico (estados y transiciones). El peso de cada uno de estos parámetros es función del dominio del problema (ej: dominios en los que hay preponderancia del modelo funcional o del modelo de datos). Para que la medición sea efectiva es necesario garantizar un nivel uniforme de particionamiento.

A fines de 1970, A.J. Albretch propone medir la funcionalidad del software a partir de la estimación de cinco items: número de entradas de datos proporcionadas por el usuario, número de salidas de datos suministradas al usuario, número de peticiones que el usuario hace al sistema en espera de una respuesta, número de archivos internos y número de archivos externos. A esta técnica se la conoce con el nombre de Punto Función

[ALB79] y el resultado se obtiene combinando los datos descriptos con un factor de peso junto a valores de ajuste de complejidad. La característica principal de esta métrica es que con los datos obtenidos y tablas que relacionan los puntos función y las líneas de código necesarias para implementarlos según el lenguaje se podrían estimar las líneas de código necesarias para implementar el desarrollo.

Una variante de la técnica de punto función es la de punto de característica [JON86] utilizada para sistemas con una complejidad algorítmica alta como por ejemplo los software para sistemas de tiempo real, de comunicaciones, de control de procesos, de simulación. En este caso se agrega un nuevo item a los cinco primitivos que tiene que ver con el número de algoritmos.

Para la *etapa de diseño* las métricas tienden a estudiar los niveles de modularización, acoplamiento entre módulos, cohesión, complejidad y tamaño de los módulos.

De Marco propone estimar la cantidad de decisiones de un módulo estudiando las estructuras de datos involucradas en el proceso. Su idea se basa en un concepto de Warnier [WAR76], quien planteó la hipótesis que la estructura interna de un módulo es isomórfica a las estructuras de datos que se mueven en torno a él.

Para la *etapa de implementación* existen métricas orientadas al tamaño, dado que se considera que el esfuerzo para escribir un programa depende en gran medida de su tamaño. La medida de tamaño clásica está dada por las líneas de código.

Dado que esta medida puede en algunos casos no ser adecuada ya que existen líneas de código más fáciles de escribir que otras, existen formulaciones alternativas midiendo el tamaño con distinto nivel de granularidad. El nivel de detalle puede ir en un extremo desde tokens (operadores y operandos básicos del lenguaje utilizados en una sentencia) hasta módulos o funciones en el otro.

Otro conjunto de métricas se basan en el estudio de las estructuras de datos. En general cualquier sistema, programa o rutina realiza algún procesamiento sobre un conjunto de datos. Algunos datos constituyen la entrada del sistema, otros son usados internamente y otros son la salida del sistema.

Los datos manipulados por una pieza de software contribuyen a la complejidad de la misma en diversas maneras. Una primer forma es a través de los datos usados (cantidad de variables), una segunda forma es según el uso de los mismos (cantidad de variables vivas que influencian la escritura de una sentencia y span de una variable definido como la cantidad de sentencias entre dos usos consecutivos de la variable), una tercer forma es vía el compartimento de datos entre componentes de programa (pasaje de parámetros o compartimento de variables globales).

Otra alternativa surge de las métricas que se basan en la estructura lógica del programa.

En este caso se asocia al programa a evaluar con un grafo de flujo que representa la estructura lógica del mismo. A partir de él pueden obtenerse medidas de la cantidad de decisiones (la hipótesis sustenta que cuanto más lineal es un programa tanto menor será su complejidad), complejidad ciclomática de Mc Cabe [MCC76] (cantidad de caminos linealmente independientes que deben encontrarse para garantizar el paso de al menos una vez por cada una de las sentencias del módulo) o nivel de anidamiento.

En la *etapa de verificación y mantenimiento* existen conjuntos de métricas que permiten valorar los defectos en el software: los números de cambios requeridos en el

diseño producto de una comprensión errónea de las especificaciones, los números de errores encontrados en el código y los números de cambios efectuados al programa para corregir los errores.

Métricas Implementadas en el Ambiente

El conjunto más importante de métricas implementadas provienen de la recolección automática desde el código fuente y son orientadas al producto de software.

Además se han contemplado una serie de métricas a ingresarse manualmente que son orientadas al proceso de desarrollo (tiempo, costo, etc). En este caso se han adoptado las etapas clásicas del ciclo de vida del software (especificación de requerimientos, diseño, implementación, prueba y mantenimiento).

Se ha definido como estrategia general el cómputo de métricas a nivel de subrutinas, para luego computar totales a nivel de módulos y a nivel de proyecto.

Métricas del producto software:

1- Métricas orientadas al tamaño:

1.1- Líneas de código: se cuentan las líneas físicas delimitadas por el terminador de línea (se distinguen líneas nulas, con comentario, con código ejecutable, con directivas al compilador, con código assembler, etc) y las líneas lógicas (distinguiendo entre sentencias ejecutables y declarativas).
Dado que existen diversas formas para contar las líneas se han utilizado en este casos las reglas propuestas por Caper Jones

1.2- Tokens: se cuentan ocurrencias de operandos y operadores desde un enfoque sintáctico. En este caso se ha trabajado sobre las métricas primitivas del Volumen de Halstead.

1.3- Procedimientos y funciones: se totalizan procedimientos y funciones en forma separada.
Esta métrica además de poder ser usada como un indicador de tamaño puede evaluar aspectos de diseño en términos de sobre o submodularización.
Un bajo grado de modularización puede estar asociado a módulos más grandes y por lo tanto más complejos.
Un alto grado de modularización puede llevar a demasiadas interfaces entre módulos y por lo tanto un mayor acoplamiento.
El tamaño promedio de subrutinas en líneas de código podría ser un indicador del nivel de modularización.

1.4- Módulos: se totaliza la cantidad de módulos fuente de un proyecto.
En este caso se puede calcular la cantidad promedio de subrutinas en un módulo. Módulos con muchas subrutinas pueden ser indicadores de archivos fuente demasiado grandes, con una sobrealocación de responsabilidades y por lo tanto más difíciles de mantener.

2- Métricas orientadas a la complejidad lógica

2.1- Complejidad ciclomática: se calcula a partir de los nodos predicados (sentencias que albergan condiciones, ej: if, while, repeat, for, case)

2.2- Nivel de anidamiento: en este caso se calcula el anidamiento promedio de la rutina y el máximo nivel de anidamiento (puesto que una gran cantidad de sentencias con bajo anidamiento escondería en promedio zonas de código complejas).
El cómputo de los niveles de anidamiento se efectúa al encontrar sentencias condicionales o repetitivas no así en el caso de bloques tipo begin-end que representan agrupamiento lógico de sentencias.

2.3- Violación de conceptos de programación estructurada: representada por rutinas que no siguen el flujo lineal de control (único punto de entrada y único punto de salida)

3. Métricas orientadas a la estructura de datos

3.1- Cantidad de variables: en este caso se ha apelado a la definición de Dunsmore contando como variables todos los identificadores excepto los nombres de procedimientos, funciones, programas y labels.

3.2- Variables vivas: se totalizan la cantidad de "usos" de variables dentro de bloques ejecutables. Con está métrica y la cantidad de sentencias ejecutables puede calcularse el promedio de variables vivas por sentencia ejecutable.

3.3- Span de variables: se calcula la suma de spans y la cantidad de spans para poder calcular el span promedio de variables según líneas físicas de código ejecutable.

4. Métricas orientadas al flujo de información

4.1- Uso de variables locales y globales: se han computado la cantidad de usos de variables globales internas (declaradas en el módulo), variables globales externas predefinidas (declaradas en un módulo no disponible para el análisis), variables globales externas no predefinidas (declaradas en un módulo disponible para el análisis).
Estas medidas en general están orientadas a obtener alguna noción del grado de acoplamiento entre subrutinas o módulos.
Teniendo en cuenta que el uso de variables globales en general puede acarrear efectos secundarios no deseados, estas medidas pueden dar una idea también del grado de propensión a error de una subrutina, módulo o sistema.
También se computan el uso de variables locales.
En algunos casos es también necesario computar los usos potenciales de variables globales (no se usan realmente pero son visibles).

4.2- Llamados a subrutinas: se computan cantidad de calls locales (a una subrutina hija), cantidad de calls globales internos (a una subrutina no hija pero dentro del módulo), cantidad de calls globales externos predefinidos (a una subrutina no hija fuera del módulo y no disponible para el análisis), cantidad de calls globales externos no predefinidos (a una subrutina no hija fuera del módulo y disponible).

4.3- Cantidad de parámetros: se cuentan los parámetros de entrada y salida. En el caso de las funciones se cuenta un parámetro adicional para el valor de retorno. Si bien el pasaje de parámetros no forma parte de los acoplamientos más graves, subrutinas con interfases muy grandes no son deseables pues tienen más propensión a errores.

4.4- Fan-in y fan-out: se calcula el número de rutinas que le pasan datos directa o indirectamente a la rutina (fan-in) y el número de rutinas que reciben datos directa o indirectamente de la rutina (fan-out)

4.5- Data bindings: se calculan la cantidad de data bindings por parámetros o por variables globales (lectura o escritura), externos o internos (con rutinas en el mismo o distinto módulo)

5- Defectos: se permite ingresar manualmente la cantidad de defectos.
Combinando está métrica con el tamaño del proyecto puede definirse el indicador:
calidad = defectos / LDC

6- Código reusado y adaptado: se permite ingresar manualmente la cantidad de líneas reusadas (código incorporado al proyecto sin adecuaciones) y cantidad de líneas adaptadas.

Métricas del proceso de desarrollo:

Se han establecido un conjunto mínimo de métricas para el proceso de desarrollo que tienden a ser un complemento de las métricas de código fuente y que permiten la combinación de ambas. Entre ellas se cuenta la posibilidad de ingresar medidas de costo, tiempo y personal involucrado.

Supongamos que un proyecto involucró P personas para producir L líneas de código en T meses a un costo de C pesos. A partir de estas métricas puede definirse:

1. Productividad = L / (P * T) => Productividad en personas-mes en términos de líneas de código

2. Esfuerzo = T * P => Esfuerzo en personas-mes

3. Costo = C / L => Costo normalizado a líneas de código

Análisis de las características del Ambiente

Los principales objetivos que se tuvieron en cuenta al diseñar la herramienta fueron:

- Ambiente integrado: se debía construir una herramienta que permitiera integrar la recolección y la muestra de métricas.
- Ambiente multilenguaje: la herramienta debía soportar la recolección y muestra de métricas provenientes de programas fuentes escritos en diferentes lenguajes
- Interfaz de usuario: la herramienta debía proveer una interfaz amigable al usuario que permitiera, entre otras cosas, realizar variados tipos de consultas para poder

analizar las métricas recolectadas. En particular debía poder presentar en forma gráfica algunos resultados.

- Formas de recolección: la herramienta debía proveer recolección de métricas en forma automática y en forma manual, y debía permitir la combinación de estas métricas.

El ambiente posee dos grandes funcionalidades: la *recolección automática de métricas* desde programas fuentes que se encargará de computar las métricas elementales y la de *interacción con el usuario* que permitirá la visualización de resultados de mediciones en forma tabular y gráfica, el ingreso de métricas en forma manual, el manejo de proyectos, etc. La estructura se presenta en la figura 1.

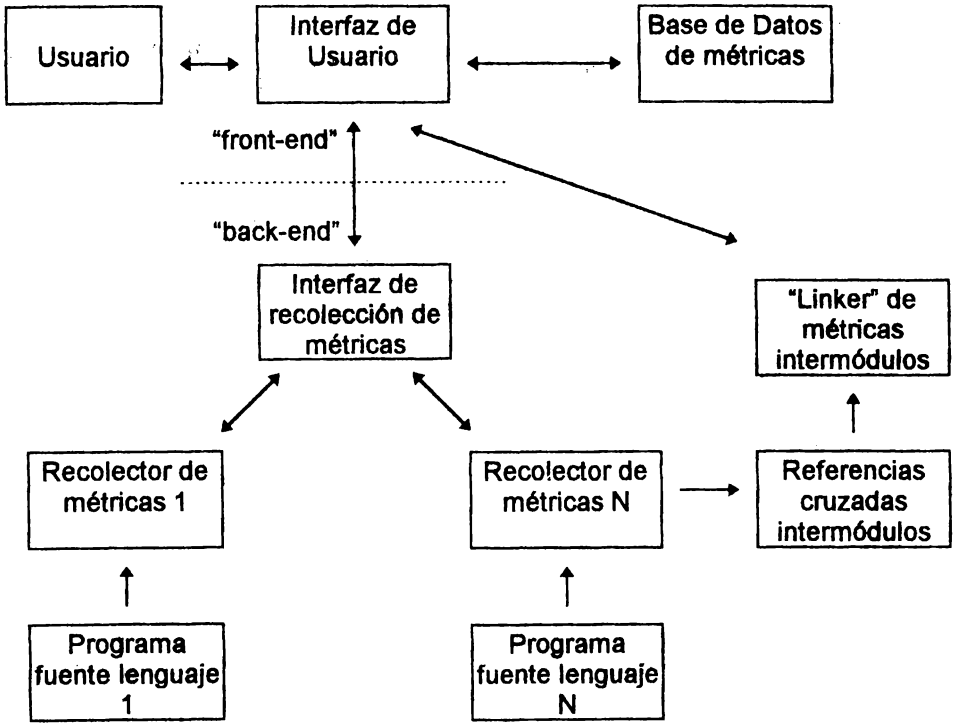

Figura 1

El módulo de recolección de métricas tiene como objetivo principal analizar el código fuente de los módulos que conforman un proyecto y computar para los mismos una serie de métricas establecidas.

Al tratarse de un ambiente de medición "multilenguaje" esta componente debe ser capaz de poder medir programas fuentes de cualquier lenguajes, poder integrar métricas intermódulos de lenguajes distintos y a la vez brindar una interfaz de servicios uniforme independientemente del lenguaje en cuestión.

Según este análisis la herramienta debe contar con tres subcomponentes: módulo de recolección dependiente del lenguaje (uno por cada lenguaje a trabajar), módulo

que resuelva las métricas intermódulos y módulo de interfaz de recolección que provee una interfaz común independiente de cada recolector.

Cuando en el ambiente se activa la función de computar métricas, la interfaz de recolección activa al recolector correspondiente, recibe los resultados de la medición y los entrega al programa cliente.

Mientras tanto, se deberá ir dejando información de referencias cruzadas a otros módulos de manera de poder computar métricas intermódulos. El módulo Linker se encargará de generar dichas métricas a partir del conjunto de referencias cruzadas.

Con este esquema tratar un nuevo lenguaje significa agregar un nuevo módulo de recolección al ambiente, sin necesidad de modificar ninguna de las componentes existentes.

El recolector implementado para protipación del ambiente ha sido para el lenguaje Borland Pascal 7.0 (exceptuando las extensiones para tratamiento de objetos) y consta de tres componentes (figura 2):

- una analizador léxico (lexer) que lee el programa fuente, reconoce tokens y computa métricas asociadas a líneas de códigos físicas y cuenta de tokens.
- un analizador sintáctico (parser) que reconoce frases válidas del lenguaje (programas, units, librerías) y computa el resto de las métricas que se resuelven a nivel de módulo, generando como salida dos archivos: uno con las métricas recolectadas (por él y por el lexer) y otro con información para el linker para computar las métricas que necesiten información cruzada de los distintos módulos.
- un linker que en base a los archivos generados por el parser (uno por cada archivo fuente del proyecto) computa el resto de las métrica que requieren información cruzada de los distintos módulos.

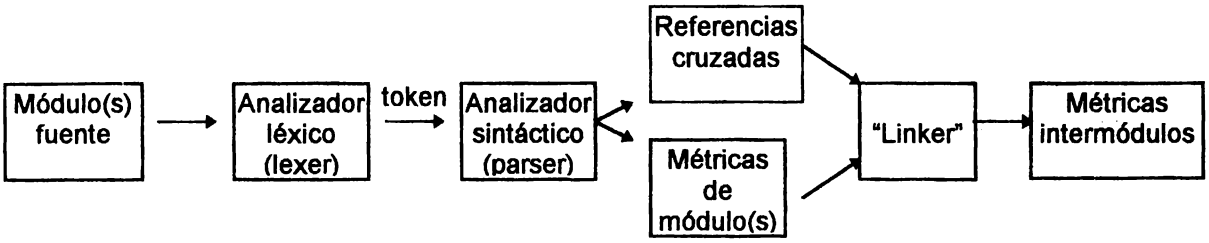

Figura 2

De las posibilidades para realizar el análisis se resolvió utilizar una solución que integrara el cómputo de las diversas métricas en una sola pasada realizando un análisis léxico-sintáctico completo. Como ayuda al proceso se utilizó la herramienta Lex & Yacc para implementar el reconocedor.

Las principales funciones de la componente de interfaz de usuario se enumeran a continuación:

- Manejo de proyectos: un proyecto está compuesto por los módulos de código fuente de la aplicación que deseamos medir. Está función permite crear, modificar y consultar proyectos de interés, que servirán de datos de entrada a las mediciones.

- Medición de módulos: esta función permite realizar mediciones iniciales o recálculos sobre módulos seleccionados y sobre todo el proyecto
- Editor de consultas: dado que uno de los objetivos al diseñar el ambiente fue la flexibilidad para realizar consultas se ha desarrollado un generador de consultas que permite al usuario construir y guardar cualquier tipo de consulta a cualquier nivel (proyecto, módulo o rutina)
- Editor de métricas: permite construir nuevas métricas como combinación de las elementales a través de expresiones aritméticas.
- Editor de gráficos: para construir gráficos comparativos entre métricas de un proyecto o de proyectos distintos.

Aspectos de Implementación

El ambiente está implementado para correr bajo la familia de sistemas operativos Windows (3.1 en adelante).

La implementación de la componente de interfaz de usuario se ha desarrollado en Visual Basic lo que permitió lograr una interfaz amigable, robusta y de buena calidad.

Las métricas recolectadas son almacenadas sobre una base de datos utilizando conectividad ODBC lo que permite usar distintos Database Management Systems (la base de datos nativa elegida fue Microsoft Access for Windows).

La implementación de la componente de recolección incluye los módulos de interfaz de recolección y linker (módulos integrantes del ambiente y son una DLL escrita en lenguaje C), y el módulo de recolección dependiente del lenguaje (debe ser una DLL escrita en cualquier lenguaje, en el prototipo se ha implementado con lenguaje C).

Este ambiente se ha desarrollado en un año de trabajo de investigación e implementación.

Líneas de Trabajo actuales

Si bien el desarrollo que se expone formó parte de un Trabajo de Grado de Licenciatura en Informática, el contexto del tema es el Proyecto Automatización de Oficinas que se desarrolla en el LIDI y que se relaciona directamente con el Magister en Automatización de Oficinas de la UNLP.

Un tema de particular interés en dicho proyecto es el de calidad de sistemas de software y en particular el gerenciamiento de proyectos según normas de calidad.

Una línea de trabajo actual es la especificación de metodologías para el desarrollo de sistemas según normas ISO 9000.

El ambiente desarrollado constituye una herramienta de evaluación útil para la recolección de datos de cada una de las fases de un proyecto de software, permitiendo realizar experiencias y comparaciones según la metodología y lenguaje utilizados.

Precisamente una línea de trabajo actual consiste en obtener conclusiones sobre sistemas desarrollados y en desarrollo en el LIDI, con diferentes paradigmas y lenguajes.

Conclusiones

Se han expuesto las características de un ambiente para evaluación de sistemas de software mediante recolección de métricas y estimaciones.

El desarrollo realizado forma parte de un trabajo de Tesina de Licenciatura en Informática y naturalmente está abierto a correcciones y mejoras. De todos modos constituye una herramienta interesante para la evaluación de proyectos de software dentro de una organización, así como para su empleo en el ámbito académico.

Toda la documentación y el sistema están disponibles en el LIDI.

Bibliografía

[ALB79] Albretch A., "Measuring Application Development Productivity", Proc. IBM Applic. Dev. Symposium, Monterey, 1979

[ART93] Arthur L., "Improving Software Quality", Wiley, 1993.

[BOE81] Boehm B., "Software Engineering Economics", Prentice Hall, 1981.

[DEM82] De Marco T., "Controlling software proyect", Yourdon Press, 1982.

[GRA87] Grady R. y Caswell D., "Software Metrics: Establishing a Company-Wide Program", Prentice-Hall, 1987.

[HUM89] Humphrey W., "Managing the Software Process", Addison Wesley, 1989.

[JON86] Jones C., "Programming Productivity", Mc Graw Hill, 1986.

[MCC76] Mc Cabe T., "A Software Complexity Measure", IEEE Trans. Software Engineering, Vol. 2, Num. 6, 1976.

[PRE93] Pressman R., "Ingeniería del software", Mc Graw Hill, 1993

[WAR76] Warnier J., "Logical Construction of programs", Van Nostrand Reinhold, 1976