Aplicaciones de Cómputo Científico: Mantenimiento del Software Heredado

Autor: Mariano Méndez

Director: Dr. Fernando G. Tinetti

"Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas "

"Facultad de Informática - Universidad Nacional de La Plata" 2015

To Fernando, for his support and friendship

To Me, because I deserve it

Índice general

Agradecimientos

Me gustaría agradecer a mi Director y amigo el Dr. Fernando G. Tinetti por el apoyo, la paciencia, la disposición y sobre todo por su amistad. Le agradezco además por darme esta gran oportunidad y por permitirme también hacer una vez más lo que más amo.

También a los integrantes de la Facultad de Informática de la Universidad de La Plata por permitir que me embarcara en este proyecto. Un agradecimiento especial va dirigido al Prof. Armando De Giusti, a la Prof. Patricia Pesado y al Instituto de Investigación en Informática LIDI.

A mi colega y amigo el Dr. Jeffrey Overbey al cual admiro por su trabajo y contribuciones al lenguaje de programación Fortran.

Al Dr. Ralph Johnson de la Universidad de Urbana-Champain, Dr. Thomas Clune del National Aeronautics and Space Administration (NASA), a Ian Chivers del ACM Fortran Forum y al Dr Ramiro Saurral de FCA de la Universidad de Buenos Aires. Además a Sergio Martín y Federico Díaz por los divertidos momentos compartidos.

A todos aquellos colegas que he conocido en todos los eventos a los que he concurrido gracias a este trabajo.

A quien alguna vez me dijera que "Nunca sería un buen científico" su comentario me incentivó aún más.

A mi familia, especialmente a mi adorada compañera de vida Melina, por todo el apoyo, la ayuda y paciencia que ha tenido a lo largo de este camino.

Gracias Abuelo José Siaskiewicz por haber sido un ejemplo para mí.

Finalmente, le doy gracias a la vida por haberme otorgado la capacidad y la oportunidad de hacer este trabajo posible.

Capítulo 1

Introducción

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." – Martin Fowler

1.1. Antecedentes y Motivación

Los Científicos han estado produciendo software desde antes que esta palabra se creara [205], de hecho el primer programa almacenado en la memoria de una computadora corrió en la ENIAC a fines de 1946 [90] así como la palabra "software" fue usada por Tukey en 1958 por primera vez en [205]. En la misma línea los modelos matemáticos, el software y los algoritmos que el software codifica desembocaron en lo que hoy llamamos Ciencia Computacional (en inglés Computational Science) [70].

Durante las últimas cinco décadas, Fortran ha sido el lenguaje de programación más utilizado para crear software científico [34, 214, 66, 120, 28, 166, 162, 158, 44, 21, 189, 58]. Este lenguaje se caracteriza por su larga existencia, y además por ser considerado el primer Lenguaje de Alto Nivel. Fortran vio la luz del día entre 1954 y 1956 [14, 12], éste incluso precede conceptos como el de "byte", forjado por John W. Tukey en [205] y al término "Ciencias de la Computación" usado por primera vez por Louis Fein en 1959 [80]. Fortran no quedó relegado del resto de los lenguajes de programación modernos, éste ha sufrido un proceso

evolutivo muy particular que ha tenido lugar durante los últimos cincuenta años, este mismo proceso lo mantiene actualizado hasta nuestros días [157]. Dicha evolución se ve reflejada en las seis revisiones de su estándar (1967, 1978, 1991, 1997, 2004, 2010). Como resultado de la misma, se han obtenido las distintas versiones conocidas del lenguaje: FORTAN66, FORTRAN77, Fortran 90, Fortran 95, Fortran 2003, y Fortran 2008 (la última especificación del estándar del lenguaje de programación) [87, 84, 9, 85, 116, 118]. Las revisiones del estándar de Fortran aspiran a mantener al lenguaje actualizado, pero durante este proceso el haber preservado una compatibilidad hacia atrás (en inglés backward) entre versiones diferentes del estándar, mantiene operativas construcciones antiguas y obsoletas del lenguaje (la instrucción GO TO, la instrucción IF Aritmético, el uso de los COMMON Blocks, etc.). Como resultado de esto, muchas operaciones pueden ser escritas de diversas formas, a veces se pueden encontrar tres o incluso cuatro formas distintas para escribir una porción de código fuente que realice exactamente la misma operación:

```
....                    ....
    DO 110 I=1,10           DO 100 I=1,10
    DO 100 J=1,10           DO 100 J=1,10
    MATRIX(I,J)=0       100 MATRIX(I,J)=0
100 CONTINUE            ....
110 CONTINUE            ....
...                     ....


....                    ....
DO I=1,10               ....
   DO J=1,10            ....
     MATRIX(I,J)=0      MATRIX(1:10,1:10)=0
   END DO               ....
END DO                  ....
....                    ....
```

Durante estos cincuenta años de evolución miles de científicos han producido

millones de líneas de código fuente escritas en Fortran, algunos de estos programas están todavía ejecutándose en ambientes de producción incluso en nuestros días. Estos programas construidos por científicos tienen una característica que los hacen interesantes: son de cómputo intensivo [99]. La mayoría de estos programas todavía están escritos en viejas versiones del lenguaje y debido al hecho de que Fortran ha evolucionado para mantener la compatibilidad hacia atrás con versiones anteriores del estándar, se pueden encontrar programas escritos en FORTRAN66 o FORTRAN77 que están todavía operativos en ciertos ambientes de producción; agregando a lo anterior, que en el software escrito desde cero, aun hoy algunos programadores utilizan versiones viejas del lenguaje. Además, ciertos programas -todavía operativos- han sido dejados como herencia por sus creadores a otros programadores, convirtiéndose en el camino en Software Heredado. De esta forma, el software se ha convertido en un medio para encapsular y transmitir conocimiento.

El software Heredado (Legacy Software) posee diversas definiciones en la bibliografía [25, 185, 182, 96, 7, 211, 207, 130], todas estas definiciones comparten conceptos en común, como por ejemplo: software que ha sido construido en el pasado, software con un alto valor económico, software complejo de utilizar, entre otros. Estos programas han podido, pueden o podrán requerir a lo largo de su existencia que se les apliquen cambios correctivos, adaptativos, preventivos o perfectivos [26]. Este tipo de tarea asociada al proceso de desarrollo de software está considerada como una de las tareas que más recursos consume en dicho proceso [220, 77] y por sí misma es considerada un desafío. En añadidura, en la mayoría de los casos trabajar con software heredado escrito en Fortran implica lidiar con código fuente desconocido y poco familiar, que generalmente está construido por personas que no están disponibles en la actualidad (para ser consultadas) y su construcción se ha hecho de forma retorcida e intrincada. A veces el espíritu con el cual el programador utiliza estos programas cae en el uso de extrañas reglas como “Si no está roto, no lo arregles”, o código programado para salir del paso, por eso este tipo de software ilegible e ininteligible, probablemente no estará listo para aprovechar las nuevas instalaciones y la potencia de cálculo de un nuevo

hardware.

En la era de los multi-cores y many-cores existen todavía muchos más programas secuenciales que están corriendo en ambientes de producción que programas paralelos. Al mismo tiempo, la cantidad de sistemas heredados crece continuamente día a día. Si bien la cantidad de programas secuenciales está creciendo, el proceso para transformar un programa secuencial en su versión paralela no está completamente desarrollado, convirtiéndose en uno de los grandes desafíos en el campo de la ciencia de la computación. En nuestros días, la mayoría de estos procesos de transformación o paralelización son realizados manualmente sin asistencia de herramientas que automaticen el proceso.

Teóricamente un "Algoritmo Paralelo depende de una simple observación que es todavía crucial: los cálculos que son independientes pueden ser ejecutados simultáneamente" [107]. Transformar programas secuenciales en paralelos, en la práctica, es un proceso muy complejo y que consume mucho tiempo, especialmente si se está trabajando con sistemas heredados escritos en Fortran. En lo que respecta al Código Fuente Heredado Fortran se requieren pasos previos a la paralelización debido al hecho que estos programas pueden contener características obsoletas y no actualizadas del lenguaje, que contribuyen a que el proceso de paralelización sea casi imposible de alcanzar. En este sentido un proceso para modernizar, actualizar y manejar código fuente escrito en Fortran es esencial. También cabe destacar la importancia de la implementación de un proceso para manejar o lidiar con código heredado, en el cual la identificación, aplicación y verificación de cambios -como por ejemplo cambios correctivos, adaptativos, preventivos o perfectivos- no sea aplicado a mano sobre el código fuente y conlleve a la aparición de errores y de consumo de tiempos de trabajo excesivos. Por ello la utilización de herramientas automáticas para llevar a cabo este tipo de tareas son preeminentes y altamente requeridas por parte de los programadores. De acuerdo con esto, los Integrated Development Environments (IDEs) o Ambientes Integrados de Desarrollo y las herramientas de análisis y transformación de código fuente deberían estar altamente integradas. Sin embargo, lo que sucede en la actualidad con los IDEs de Fortran y estas herramientas, es justamente lo contrario, no son

muy utilizados por los programadores Fortran y además no existe integración alguna entre ellas.

Este trabajo de investigación está enfocado en las Aplicaciones Científicas Heredadas escritas en Fortran por diversos motivos que a continuación se enumeran:

1. Existe cierta discordancia, desajuste o diferencia en el proceso de desarrollo de este tipo de software [121, 21].

2. El software es de cómputo intensivo [99].

3. La forma en que el proceso evolutivo sufrido por el lenguaje y su compatibilidad hacia atrás actuó sobre el software escrito en Fortran [157].

4. La etapa de mantenimiento es por lejos la más costosa de todo el proceso de desarrollo [220, 77].

5. Existe una gran cantidad de software científico heredado escrito en Fortran disponible para ser estudiado.

En este trabajo se propone inicialmente un proceso cuyo objetivo es manejar y aplicar mejoras o modernizar software, a partir de la ejecución de una serie de pasos de forma tal que éstos permitan lograr la aplicación de tales cambios. El proceso propuesto puede ser aplicado por única vez en el código fuente o bien aplicado como un proceso iterativo e incremental. Éste se divide a su vez en etapas diferentes: la etapa de comprensión, la etapa de transformación, la etapa de verificación y la etapa de retroalimentación. Cada una de estas etapas está cubierta por un conjunto de pasos, que conforman un flujo de trabajo, perteneciente a dicho proceso. Este proceso ha sido concebido para brindar un marco de trabajo teórico para el manejo de cambios, correcciones y/o mejoras al software heredado escrito en Fortran. También, dicho proceso podría ser utilizado en la construcción de software desde cero. Además como segunda componente de investigación, nos hemos enfocado en la construcción de un amplio conjunto de herramientas de análisis de código fuente completamente integradas a un Ambiente de Desarrollo de Software (IDE). Algunas de estas herramientas son: Construcción del Árbol

Estático de Llamadas, Análisis de Áreas de COMMON BLOCKS, un Localizador de Características Obsoletas / en Desuso del Lenguaje, etc. Estas herramientas han sido construidas con el objetivo de contribuir con los programadores para que éstos rápidamente puedan adquirir conocimiento sobre el código fuente sobre el cual están trabajando. Un tercer componente de esta investigación se ha enfocado en la construcción de un conjunto de transformaciones automáticas de código fuente Fortran con el propósito de mejorar, actualizar y modernizar la estructura interna del software sin cambiar la funcionalidad original del mismo. Otro componente en el cual se ha centrado la investigación se basa en la etapa de verificación, para ello se ha construido una herramienta que integra los contadores de hardware en el código fuente en el cual se está trabajando. Esta integración permite a los programadores insertar automáticamente el código fuente necesario para realizar mediciones usando las librerías PAPI (Performance Application Programming Interface). Este tipo de información faculta a los programadores a encontrar puntos en el código fuente que pueden estar actuando, por ejemplo, como cuellos de botella de cómputo. Por otro lado, estas mediciones pueden proveer información sobre las mejoras que se han realizado al código fuente tras la aplicación de transformaciones sobre el mismo. Finalmente, se ha implementado e integrado un conjunto ampliamente conocido y utilizado de métricas de software como por ejemplo: Complejidad Ciclomática [146], las métricas de Halstead llamadas Software Science Metrics [104, 103, 105], y el índice de mantenibilidad de Coleman [54] entre otros. Todas estas métricas han sido integradas en el IDE llamado Eclipse con el objetivo de brindar información sobre código fuente a los programadores "on the fly" que significa "mientras ellos están programando". El enfoque usado para la construcción e implementación del conjunto de todas las herramientas presentadas en esta tesis está basado en el mismo enfoque que utilizan los compiladores, que consta del recorrido y/o modificación de árboles sintácticos abstractos (Abstract Syntax Trees) [6]. Por tanto, en la búsqueda de una respuesta integral al problema de la modernización del software heredado escrito en Fortran, con un proceso definido mediante la utilización de herramientas automáticas integradas en un IDE moderno, se proponen las siguientes contribuciones y

objetivos.

1.2. Objetivo

El principal objetivo de este trabajo es **explicar e identificar la naturaleza y las cualidades esenciales de un enfoque (proceso, técnicas y herramientas) para identificar y aplicar cambios automáticos en código fuente heredado escrito en Fortran, especialmente software científico, con el propósito de modernizar la estructura interna del software sin modificar su funcionalidad original**. Objetivos secundarios:

- Estudiar el estado del arte.
- Proponer un proceso bien definido para manejar cambios en el software científico heredado escrito en Fortran, haciéndolo extensible a todo el software heredado escrito en Fortran.
- Implementar un set de herramientas automatizadas capaces de extraer información significativa y conocimiento del código fuente heredado escrito en Fortran para ser utilizado por los programadores.
- Implementar un conjunto de herramientas automatizadas para aplicar en forma automática transformaciones de código fuente en programas escritos en Fortran de programas desactualizados con el objetivo de obtener una versión actualizada del mismo sin cambiar su funcionalidad.
- Implementar un conjunto de métricas que sean calculadas automáticamente mientras el software está siendo desarrollado con el objetivo de obtener información valiosa sobre el mismo.
- Implementar una herramienta para integrar bibliotecas de contadores de hardware como las PAPI con el objetivo de obtener información dinámica sobre el cómputo del software.
- Aplicar nuestro enfoque a software científico heredado escrito en Fortran.

1.3. Publicaciones Científicas

Esta tesis ha sido desarrollada y ha generado hasta el momento las siguientes publicaciones científicas:

- Mariano Méndez, Fernando G.Tinetti. **First Steps Towards a Tool for Legacy Systems**. XVII Congreso Argentino de Ciencias de la Computación (CACIC2011). Isbn 978-950-34-0756-1 pp. 799-808 La Plata, Buenos Aires, Argentina, Octubre 12-15, 2011.

- Fernando G.Tinetti, Mariano Méndez, Armando De Giusti. **Restructuring Fortran Legacy Applications for Parallel Computing in Multiprocessors**. Journal of Supercomputing. Springer. 2013. V 63(1).

- Mariano Méndez y Fernando G. Tinetti. **Aplicaciones Científicas Numéricas: El (Ciclo de Vida del) Software Heredado**. XIV Workshop de Investigadores en Ciencias de la Computación (WICC2012). p. 523-527, ISBN: 978-950-766-082-5.

- Fernando G. Tinetti and Mariano Méndez, **Fortran Legacy Software: Source Code Update and Possible Parallelization Issues**. ACM Fortran Forum. 31(1), 5-22, Abril 2012)

- Mariano Méndez, Jeffrey Overbey, and Fernando Gustavo Tinetti. **Legacy Fortran Software: Applying Syntactic Metrics to Global Climate Models**. In XVIII Congreso Argentino de Ciencias de la Computación. 2012.

- Fernando G. Tinetti, Sergio M. Martín, Fernando E. Frati, Mariano Méndez. **Optimization and Parallelization Experiences Using Hardware Performance Counters**. 4to. Congreso Internacional de Supercómputo en México (ISUM 2013), abstract publicado en el programa general, pág. 33.

- Fernando G. Tinetti, Mariano Mendez. **An Automated Approach to Hardware Performance Monitoring Counters**. International Confe-

rence on Computational Science and Computational Intelligence (CSCI'14). March 10-13, 2014, Las Vegas, USA.

- Mariano Méndez, Fernando G. Tinetti. **Integrating Software Metrics for Fortran Legacy into an IDE**. XI Workshop de Ingeniería de Software (CACIC 2014). San Justo, Buenos Aires, Argentina. Octubre 2014.

- Méndez, M., Tinetti, F. G., and Overbey, J. L. (2014, November). **Climate Models: Challenges for Fortran Development Tools**. In Proceedings of the 2nd International workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (pp. 6-12). IEEE Press.

1.4. Estructura

La estructura de esta tesis está organizada de la siguiente forma, en el capítulo 2 se realiza una descripción del enfoque teórico y una revisión del estado del arte. En el capítulo 3 se presenta y describe un proceso de desarrollo guiado por el cambio. Luego en el capítulo 4 se describe la etapa de comprensión planteada para el proceso de desarrollo. En el capítulo 5 se describe la etapa de transformación conjuntamente con la implementación de herramientas a ser utilizadas en la misma. Posteriormente en el capítulo 6 se describe la etapa de verificación que forma parte del proceso de desarrollo guiado por el cambio y se presentan herramientas automatizadas para ser utilizadas en la misma. En los capítulos 7, 8 y 9 se aplica el proceso propuesto a ejemplos concretos. Finalmente en el capítulo 10 se detallan las conclusiones. Por último se presentan tres anexos, el anexo A y el B contienen el código fuente de cada una de las iteraciones del proceso, los resultados de cada corrida, etc. El último anexo C contiene código fuente variado.

Capítulo 2

Foco Teórico y Literatura Relacionada

En el presente capítulo se realizará un análisis sobre el foco teórico en el que se sustenta este trabajo de investigación. El análisis abarcará al Software Científico, la evolución de Fortran, el Software Heredado, las áreas vinculadas de HPC (High Performance Computing) y de la Ingeniería de Software. A su vez se realizará un análisis de los trabajos relacionados con el tema de investigación.

2.1. El Software Científico

En los albores de la Ciencia de la Computación, cuando su nombre no había sido todavía forjado, toda la producción de software era software científico. Algunos ejemplos destacados son: el primer programa almacenado en una computadora [186], las simulaciones computacionales de explosiones nucleares [208], el cálculo de trayectorias balísticas [62], el programa de Lehmer [40], el trabajo inicial de Von Neumann y su grupo de investigación que ejecutó el primer pronóstico climático en una computadora (ENIAC) Electronic Numerical Integrator And Computer (Computador e Integrador Numérico Electrónico), en 1949 [74, 143] entre otros.

Una característica destacable que tenía la arquitectura de ENIAC entre 1945

y 1949 era la "programación directa" que le otorgaba capacidades paralelas a la máquina [90]:

"*When one adds the complexity of incorporating ENIAC's built-in capability to perform parallel operations, it is remarkable that any degree of success was ever achieved during this first period. (Anyone now doing research in parallel computing might take a look at ENIAC during this first time period, for indeed ENIAC was a parallel computer with all of the problems and opportunities that this entails.)* ".

En 1947 Clippinger y Von Neumann (entre otros) realizaron ciertas modificaciones arquitecturales a ENIAC abriendo una nueva etapa en la vida de la misma, el "segundo período de la historia de ENIAC (1948-1955)". Fue aquí y con estos cambios que Von Neumann, ahora conocido por "First Draft of a Report on the EDVAC" en el cual describe el nuevo concepto lógico de las máquinas y lo que en nuestros días llamamos la arquitectura de Von Neumann [97], que adquirió gran relevancia en las Ciencias de la Computación.

El desarrollo de las Ciencias de la Computación está fuertemente asociado y ligado al desarrollo y la evolución de otras ciencias. Un ejemplo preponderante ligado a Ciencia de la Computación lo vemos en las Ciencias de la Atmósfera. El nacimiento de la meteorología moderna y la predicción del clima puede ser rastreado hacia los inicios del siglo XX [74, 215, 143]. En 1904 Vilhem Bjerknes publicó un artículo en el cual propuso un procedimiento y un conjunto de ecuaciones que permitían a los científicos predecir el clima. Este conjunto de siete ecuaciones independientes describían el comportamiento de siete variables: temperatura, presión, densidad, las tres componentes de la velocidad y la humedad, ver Figura 2.1. Además propuso un proceso de dos pasos para obtener una predicción racional: un paso llamado diagnóstico y un paso llamado pronóstico. Las soluciones a la ecuación de Bjerknes requerían una gran cantidad de cálculos matemáticos y dada la falta de herramientas automáticas que lograsen tales cálculos, en esa época, obstaculizó la demostración que validara que este modelo propuesto fuese correcto.

Al inicio de los años 20 la idea de dividir el espacio en una grilla compuesta por celdas, y cada celda con sus propios valores para las diferentes variables (tem-

$$\frac{f}{m} = \frac{1}{\rho}\frac{dp}{dx}$$

$$f_r = \frac{f}{a}\frac{1}{\rho}\mu(\nabla \cdot (\mu\nabla v) + \nabla(\lambda\nabla \cdot v)))$$

$$\frac{dv}{dt} = -(1/\rho)\nabla p - g(r/r) + f_r$$

$$g = g_e$$

$$c_v\frac{dT}{dt} + p\frac{d\alpha}{dt} = q + f$$

$$\frac{d\rho}{dt} + \rho\nabla \cdot v = 0$$

$$p = \rho RT$$

Figura 2.1: Las Ecuaciones de Bjerknes

peratura, presión, densidad, humedad y las tres componentes de la velocidad) y la aplicación del método de diferenciación finita para resolver las ecuaciones diferenciales, convergieron en el primer "Numerical Weather Prediction" propuesto por L. F. Richardson. Mientras que las técnicas de Richardson hacían un uso simplificado del conjunto de ecuaciones de Bjerknes la cantidad de cálculos requeridos para obtener el pronóstico consumía todavía mucho tiempo. Con el objetivo de acelerar este proceso Richardson imaginó lo que él llamó "The Forecast Factory" una versión vanguardista de un proceso de cálculo colaborativo y automatizado. Éste quería tomar a unas 64 mil personas cada una de ellas equipadas con un calculador mecánico para ejecutar una pequeña porción del cálculo total, ver Figura 2.2, esto puede ser visto como una temprana visión de lo que hoy llamamos "Crowd Computing" [74, 215].

Después de la Segunda Guerra Mundial, John Von Neumann reunió a un grupo de científicos climáticos para fundar el Metheorology Project. Von Neumann auspiciaba el desarrollo del modelado del clima "él esperaba que el modelado del clima pudiera desembocar en el control del mismo" [74]. Este grupo de científicos liderado por Jule Charney, corrió en una computadora el primer pronóstico regional automatizado del clima en 1949 [74, 215, 143].

Figura 2.2: Richardson's "The Forecast Factory"

Esta simulación dividía América del Norte en una grilla de 270 puntos y cada uno de estos puntos estaban separados por 270 Km. Éste fue el primer pronóstico del clima de 24 horas cuyos cálculos tardaban 24 horas en ser realizados, ver Figura 2.3. El próximo paso fue la construcción de un modelo climático de toda la atmósfera de la Tierra. Este primer modelo de circulación general fue corrido en mayo de 1955 por Norman Phillips. Era un modelo cuasi geotrópico de dos niveles, fue la primera simulación atmosférica a larga escala ejecutada en una computadora. El modelo usaba una grilla de 272 puntos que corrió en el Mathematical Analyzer, Numerical Integrator, and Computer (MANIAC I) [143].

Es importante destacar que muchas otras áreas de aplicaciones han tenido una relación similar a la de las Ciencias de la Atmósfera con las Ciencias de la Computación que se han descrito en los párrafos anteriores, aunque en algunos casos han tomado menos notoriedad. En el caso del software relacionado con el petróleo, por razones estrictamente económicas o en el caso del software relacionado con la energía nuclear, por ser directamente estratégicas, son otros ejemplos de áreas relacionadas con las Ciencias de la Computación.

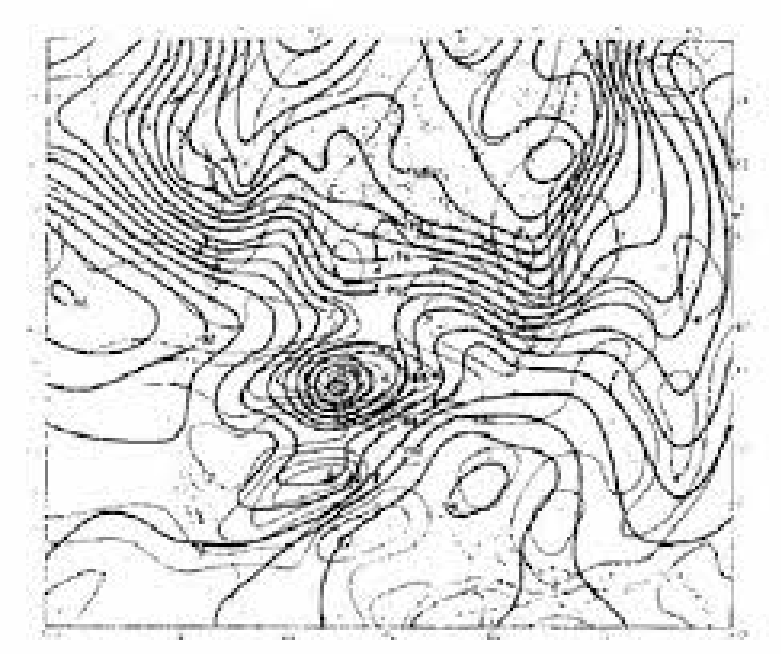


Figura 2.3: Primera predicción climática hecha por ENIAC

2.2. Fortran

Fortran es unos de los lenguajes de programación de alto nivel más antiguos que ha sido utilizado por los científicos para producir software desde sus orígenes [157, 13, 151]. Podemos decir que a través del tiempo, éste se ha convertido en el lenguaje de programación científica "De Facto" o por lo menos en el más utilizado [214, 17, 120, 158, 44, 21, 189, 19]. La primera versión del manual del usuario de este lenguaje de programación fue publicada el 15 de octubre de 1956, según [13] por el equipo de IBM liderado por J. Backus. A través de su extensa vida este lenguaje de programación ha atravesado y sufrido un particular proceso evolutivo. Éste ha sido el primer lenguaje de programación de alto nivel en ser estandarizado [86, 187].

2.3. La Evolución de Fortran

Desde su nacimiento se conocen 10 versiones del lenguaje de programación FORTRAN I, FORTRAN II, FORTRAN III, FORTRAN IV, FORTRAN 66,

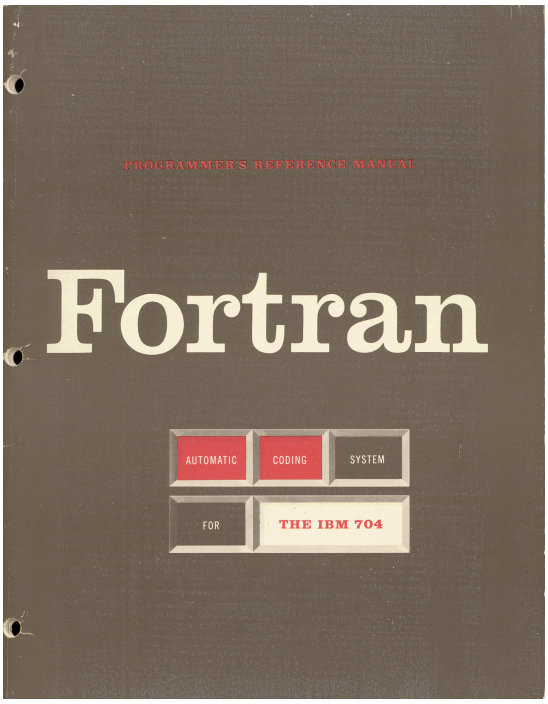


Figura 2.4: El Manual de FORTRAN I para la IBM 704

FORTRAN77, Fortan 90, Fortran 95, Fortran 2003, Fortran 2008 y posiblemente dentro de poco tiempo se publique Fortran 2015 cuya versión está siendo revisada en estos días. Desde que Fortran vio la luz del día (los últimos 60 años) el lenguaje de programación ha sufrido una serie de revisiones, un grupo de ellas se han convertido en estándares (1966, 1978, 1991, 1997, 2004 y 2010). Algunas han sido consideradas revisiones mayores (1966, 1978,1991 y 2003) y otras se consideran revisiones menores (1995 y 2008) del estándar; debido a este hecho se considera que el lenguaje ha pasado por "Siete Eras" de evolución [157].

La Primera Era: Los Orígenes FORTRAN I, FORTRAN II, FORTRAN III

Fortran fue descripto por primera vez el 15 de octubre de 1956 por J. Backus en "IBM Programmer's Reference Manual, the Fortran Automatic Coding System for the IBM 740". Esta versión del lenguaje fue conocida como FORTRAN I y contaba con un conjunto de tan solo 32 instrucciones [14, 12], ver Figura 2.4 .

Tras un período de testeo del lenguaje fueron encontradas algunas fallas en lo referido a su diseño, por ello los creadores del mismo empezaron a trabajar

en una nueva versión. Este hecho, está documentado en un artículo publicado en septiembre de 1957 intitulado "Proposed Specifications for FORTRAN II for the 704". Esta versión agregaba nuevas características al lenguaje [56] como ser: nuevos tipos de datos, la posibilidad de crear subrutinas y funciones, áreas comunes de memoria (COMMON BLOCKS), ente otras. En 1958 se dio a conocer la nueva versión de este lenguaje de programación diseñado por Nelson, Ziller y Backus. En el mismo período que FORTRAN II había sido publicado, IBM ya estaba trabajando en FORTRAN III, una versión de Fortran que ni siquiera logró alcanzar el estatus de producto [13].

La Segunda Era: FORTRAN IV y FORTRAN 66

A pedido de los usuarios de IBM una nueva versión de Fortran fue publicada en 1961, llamada FORTRAN IV. Esta versión aportaba nuevas características y mejoras a las anteriores. Una de estas características fue la supresión de las dependencias que las instrucciones de entrada / salida (I/O) tenían respecto a la arquitectura de las computadoras para la que era diseñada. Por ello se produjeron compiladores de FORTRAN IV para equipos como IBM/360 Mainframe,IBM 7090/7094 y otros.

Hacia mayo de 1962, el comité ANSI (American National Standards Institute) comenzó a trabajar en el primer estándar de Fortran. Este comité, formado por académicos y expertos de la industria derivó en dos estándares. El primero, basado en FORTRAN IV, que se convertiría en un hito en la historia de los lenguajes de programación, famosamente conocido como FORTRAN 66, en el cual todas las dependencias arquitecturales de un equipo en particular fueron eliminadas. El segundo, basado en FORTRAN II, llamado Basic FORTRAN en el cual estas dependencias arquitecturales no habían sido eliminadas. El estándar publicado en 1966 fue el primer estándar de un lenguaje de alto nivel publicado en la historia de la programación. FORTRAN 66 incluía [87]:

- Instrucciones de Control:

    - MAIN PROGRAM, SUBROUTINE, FUNCTION, y BLOCK DATA program units.
    - DO loops.
    - IF lógico e instrucción IF aritmético.
    - Instrucción FORMAT .
    - Instrucciones: CALL, RETURN, PAUSE, y STOP.
    - Instrucción GOTO.
    - Instrucción GOTO asignado
    - Instrucción GOTO computado.
- Instrucciones de Entrada y Salida:
    - READ, WRITE, BACKSPACE, REWIND, y ENDFILE.
- Instrucción de Asignación.
- Instrucciones de Especificación:
    - Instrucción COMMON.
    - Instrucción DIMENSION.
    - Instrucción EQUIVALENCE.
    - Instrucción DATA para la especificación de valores iniciales.
- Tipos de Datos:
    - INTEGER, REAL, DOUBLE PRECISION, COMPLEX y LOGICAL.
- Otras Características:
    - Funciones intrínsecas y externas.
    - Constantes Hollerith en las instrucciones DATA y FORMAT, y actuando como parámetros actuales en los procedimientos.
    - Más de seis caracteres para los nombres de los identificadores.
    - Comentarios.

La Tercera Era: FORTRAN 77

Hacia 1969 el comité ANSI vio la necesidad de realizar una revisión al estándar de FORTRAN 66. Debido a que el lenguaje había sido ampliamente utilizado y aceptado en los últimos años los vendedores de compiladores habían creado diferentes dialectos del mismo lenguaje, introduciendo en cada caso distintas características al lenguaje. Por estos motivos, en 1978 el American National Standards Institute publicó una nueva versión del estándar (ANSI X3.9-1978) conocida históricamente como FORTRAN77. Hacia 1980 la International Standard Organization (ISO) lo adoptó como un estándar internacional (IS 1539:1980). Esta versión del estándar de Fortran fue históricamente la más utilizada y ampliamente conocida.

Hasta ese momento el concepto "uso desaconsejado" (deprecated) no había aparecido en el estándar ANSI, sólo un conjunto de viejas características del lenguaje fueron eliminadas del estándar. Ver el apéndice A2 [84]. Estas características fueron:

- Datos y Constantes Hollerith :

  LINE=16HTODAY'S DATE IS:

- Leer en H (Hollerith field) como descriptor de la instrucción FORMAT.

- Indexar más allá de los límites de un arreglo.

  DIMENSION B(9,5)

  J= B(10,1)

- Transferir el control dentro de un ciclo DO.

En ese mismo año el Departamento de Defensa de los Estados Unidos creó una extensión del lenguaje llamada MIL-STD-1753, donde se añadieron algunas características nuevas al lenguaje: las instrucciones DO WHILE y END DO, la instrucción INCLUDE, IMPLICIT NONE ()una variante de la instrucción IMPLICIT) y funciones intrínsecas de manipulación de bits.

La Cuarta Era: Fortran 90

De alguna manera las decisiones tomadas por el Comité de Revisión del estándar determinaron la forma en la cual el lenguaje evolucionaría. El nombre mismo del lenguaje cambió de escribirse en mayúsculas FORTRAN a escribirse como Fortran. Como se puede apreciar en un fragmento del estándar:"Note that the name of this language, Fortran, differs from that in FORTRAN 77 in that only the first letter is capitalized. Both FORTRAN 77 and FORTRAN 66 used only capital letters in the official name of the language, but Fortran 90 does not continue this tradition ...". Este detalle al parecer menor, denota cuan rígidas eran las características de los estándares anteriores. Ésta fue una de las mayores revisiones que tuvo el lenguaje, en ella se introdujeron todas aquellas instrucciones faltantes para poder utilizar el paradigma de la programación estructurada. Muchas de las características que tenía la extensión creada por el Departamento de Defensa de los Estados Unidos fueron agregadas a esta nueva revisión. Otro importante cambio fue la introducción del formato libre en la escritura del código fuente, que hasta ese momento poseía un formato fijo de escritura. Esta revisión mayor del estándar es conocida como Fortran 90 [85].

A partir de este punto el lenguaje de programación tenía características modernas como los lenguajes C o Pascal, pero conservaba características antiguas, que se definieron en el primer estándar o incluso previas al mismo . Estas características serían marcadas como obsoletas. A razón de esto, en el apéndice B.1 dice: "The list of deleted features in this standard is empty". Por ende un programa que cumplía con el estándar de FORTRAN 77 también lo hacía con el estándar de Fortran 90. La lista de características marcadas como obsoletas es la siguiente [85]:

1. Alternate return.
2. Instrucción PAUSE.
3. Instrucción ASSIGN.
4. Instrucción GO TO asignado.

5. Instrucción IF aritmético.
6. Variable de control de la instrucción DO de tipo Real y de doble precisión.
7. Shared DO loop termination y terminación en una estructura que no sea otra que END DO or CONTINUE.
8. Salto a un END IF por fuera de un bloque IF.
9. Especificadores Assigned FORMAT.
10. Descriptor cH.

La Quinta Era: Fortran 95

Si bien esta revisión del lenguaje propuso cambios menores fue la primera que introdujo la eliminación de algunas características marcadas como obsoletas en el estándar anterior. Éstas son [115]:

- Variables de tipo double y real en los índices de los DO.
- Saltar a una instrucción END IF desde fuera del bloque de la instrucción IF.
- La instrucción PAUSE.
- Las instrucciones ASSIGN y assigned GO TO.
- Los descriptores H.

A pesar de que dichas características marcadas como obsoletas fueron borradas, los fabricantes de compiladores mantuvieron la compatibilidad hacia atrás: “Unlike Fortran 90, Fortran 95 was not a superset; it deleted a small number of so-called obsolescent features. This incompatibility is more theoretical than real however, as all existing Fortran 95 compilers include the deleted features as extensions"[53].

La Sexta Era: Fortran 2003

Sin pausa pero sin prisa la modificación del estándar de Fortran continuaba, en el 2004 fue publicada otra revisión mayor llamada Fortran 2003. Esta revisión introducía al lenguaje la capacidad de poder utilizar el paradigma de la Programación Orientada a Objetos (conceptos tales como herencia, polimorfismo, etc.). Se introdujeron mejoras en lo que respecta a manipulación de datos, operaciones de entrada y salida entre otros [116, 183]. Según Ian Chivers y Jane Sleightholme no hay un compilador Fortran 2003 que tenga todas las características del estándar habilitadas de Fortran 2003 [50] (a la fecha de ser consultado). Entre los compiladores que más características implementan de Fortran 2003 son aquellos de las empresas: Intel, Cray, Pgi.

La Séptima Era: Fortran 2008

Teniendo en cuenta que la construcción de compiladores de Fortran 2003 no se ha producido rápidamente el comité de estandarización de Fortran igualmente consideró necesario realizar una nueva revisión, en este caso menor del mismo. Las características más destacables de esta nueva versión del estándar publicada a fines de 2012, fueron los co-arrays y los DO concurrent [157, 118].

La Evolución de Fortran

Fortran ha sobrevivido a 60 años de cambios en el seno de la Ciencia de la Computación, éste se ha ido adaptando a las necesidades de los programadores según las épocas y la tecnología. Si bien dentro de la cultura "popular" de las Ciencias de la Computación se cree que Fortran es un lenguaje obsoleto, vetusto y en desuso, su historia y su evolución demuestran todo lo contrario, dejan entrever a un lenguaje dinámico y adaptado a las necesidades de sus programadores. Que además para lograr dicha característica adoptó un peculiar proceso evolutivo ya sea, formalmente mediante los estándares, o pragmáticamente mediante la industria. Este proceso ha logrado mantener la compatibilidad hacia atrás a tal

grado de poder compilar en la actualidad un programa FORTRAN 66 y poder realizar dicha compilación con cualquier compilador moderno disponible.

2.4. El Software Heredado

No existe una única definición de "Software Heredado" (Heritage Software) también conocido como "Software Legado" (Legacy Software), a continuación se tratará de analizar y estudiar las definiciones que se encuentran en la bibliografía. En primer lugar, se analiza la etimología de "legado" y "heredado". El primer término, Software Heredado, deriva de la traducción del inglés de "Heritage Software". La raíz de la palabra heritage deriva del francés "heriter" que a su vez deriva del latín "hereditare", que en español deriva en la palabra "heredar" cuyo significado según la Real Academia Española es "Recibir de alguien algo que éste ha usado antes". Por otro lado el término "Software Legado" deriva de la traducción del término inglés "Legacy Software". La raíz de esta palabra inglesa "legacy" proviene del francés "legacie" que a su vez deriva del latín "legatum" cuyo significado es "legado", según el diccionario de la Real Academia Española, uno de sus significados es "Aquello que se deja o transmite a los sucesores, sea cosa material o inmaterial". Por ende podemos pensar en el Software Heredado o Software Legado como software hecho por terceras personas que es dejado a sus sucesores.

Existen varias definiciones de lo que es el Software Heredado, pero no hay ninguna que sea adoptada como la única definición formal.

1. Brodie and Stonebraker han definido un Sistema Heredado desde la perspectiva de los sistemas de información: "*Any information system that significantly resists modification and evolution to meet new and constantly changing business requirements.*" [35].

2. Nicolas Gold, resume el concepto de software heredado como: "Legacy Software is critical software that cannot be modified efficiently." [96].

3. Michael Mahoney propone en [144]: "Legacy software is not just old code, but rather a continuing enactment, an operative representation, of the

domain knowledge and practice embodied in it. That may explain the difficulties software engineers have experienced in upgrading and replacing older systems".

4. En [218] se define legacy software como:"legacy software is a program that is still well used by the community but was developed years ago. Many functions of legacy software have been modified and corrected over the years".

5. Zaidman en [219] define:"Legacy software is all-around: software that is still very much useful to an organization, quite often even indispensable, but a burden nevertheless. A burden because the adaptation, integration with newer technologies or simply maintenance to keep the software synchronized with the needs of the business, carries a cost that is too great. This burden can even be exaggerated when the original developers, experienced maintainers or up-to-date documentation are not available".

6. "Legacy software is a software that is hard to change and is still alive (operational), an indication that its users are still in business"[126].

7. "Legacy software is referred to software systems which were developed some time ago, which have been undergone considerable modification and which are still indispensably used nowadays. Two main features usually charaterise a legacy software system: (1) huge volume of software code, and (2) deteriorated software documentation"[139].

8. Una definición bastante simplificada de lo que es el software heredado la encontramos en [132] "The legacy software is the software that has been created or modified previously by people".

9. "Legacy software is valuable software that you have inherited. The fact you have inherited it may mean that it is somewhat old-fashioned" [63].

10. Quizá la más radical de todas las definiciones de lo que es el software heredado sea la de Michael Feathers [79]:"The main thing that distinguishes legacy code from non-legacy code is a lack of comprehensive tests".

Esta última definición de Michael Feathers si bien puede parecer radical se acerca a los procesos de desarrollo de software, como por ejemplo, los procesos ágiles de desarrollo en los cuales el testing es una herramienta fundamental a ser utilizada en dichos procesos. Además, teniendo en cuenta que el proceso en el cual el software heredado consume más recursos es el de mantenimiento donde no sólo son aplicados cambios correctivos sino también preventivos, perfectivos e incluso modificación de su funcionalidad, es coherente que aquel software que no posea tests no pueda reflejar cuáles serán los impactos de esos cambios.

Muchas de estas definiciones describen al software heredado con conceptos comunes entre ellas:

- Software de valor.
- Software desarrollado en el pasado.
- Software aún operativo.
- Software indispensable.
- Software que ha sufrido cambios a lo largo de su vida.

Existen autores que además dividen entre el concepto de software heredado y el de sistema heredado haciendo claras diferencias entre ambos, Carole Brooke y Magnus Ramage en [36] claramente distinguen estas diferencias que según ellos existen entre ambos conceptos:

- Software Heredado :"Legacy software is critical software that cannot be modified efficiently. In other words, it is software perceived by the business to be critical to its operations, and yet difficult to modify without incurring great expense (in terms of time, skill, etc). Legacy software is often described as being any or all of the following: large, old, heavily-modified, difficult to maintain, old-fashioned. Cynics reading the popular computing press might suppose that legacy software is a term invented by late capitalism, intended to make perfectly good software look outdated and thus sell more products."

- Sistemas Heredados: “In contrast, a legacy system refers to much more than the software. It is a wider system of which the software is merely a part. Other components of the system might include: people, expertise, hardware, data, business processes, and approaches to software maintenance and development. Understanding a legacy system also requires taking account of its relationship to the business environment. All these things – and especially their interactions with each other – constitute a legacy system. Thus, legacy systems consist of much more than just a technical dimension: they encompass issues of organisational structure, strategy, process, and workflow. This is a much more complex consideration.”

Teniendo en cuenta esta separación entre sistema heredado y software heredado las distintas definiciones de sistemas heredados encontradas en la bibliografía se listan a continuación:

- Visaggio et al. [207] : “A legacy system is generally one of an organization’s assets with a high economic value. Even when it ages, it is both difficult and risky to replace it. Difficult, because the system is used by many people within the organization and its replacement would involve retraining all the users to understand the new system. Risky, because the construction or purchase of a new system may go over budget and disrupt planned schedules; moreover, the new system may lack some functions the users of the previous system were used to having”.

- Paul Robertson define a los sistemas heredados en [185] como: “Typically, a legacy system is an in-place structure that is neither optimal for modern needs nor modifiable for project purposes. In dealing with legacy systems, it is important to understand the forces that govern their existence. Legacy systems for the most part were developed by a previous generation of developer—hence the term legacy. That the code is often voluminous and hard to modify is of little interest to us in this instance. The crucial issue with legacy systems is that they are generally wired into the running of a

business in a very substantial way. The security of the legacy systems is of utmost importance ".

- Otra definición se la encuentra en [76]:"But the code of a legacy system is usually a very poor expression of its design. It includes "dead" or obsolete code and "glue" code of incremental updates".

- Tambien Dillman propone una definición [68]: "A legacy system is an existing system within a company that continues to be used despite being out-of-date, incompatible with current systems, or otherwise undesirable. Generally, legacy systems tend to be difficult and expensive to modify and unnecessarily large".

- Kotogiannis considera [125] que :"A legacy system is an operational, large-scale software system that is maintained beyond its first generation of programmers. It typically represents a massive economic investment and is critical to the mission of the organization it serves".

- K. Benneth propone: "*large software systems that we don't know how to cope with but that are vital to our organization.* "[25].

- Según Hollander [108]: "According to the Free On-Line Dictionary Of Computing (FOLDOC), legacy system is defined as: "A computer system or application program which continues to be used because of the prohibitive cost of replacing or redesigning it and despite its poor competitiveness".

A medida que se profundiza en la bibliografía se puede ver que hay algunos temas que están ligados al software y los sistemas heredados:

- Escasez de documentación [181, 95].

- La complejidad y riesgo de modernizar software heredado [61, 33, 51, 55, 160].

- El costo y el esfuerzo que se requieren para mantener este tipo de software [211, 69, 111, 175, 188].

Actualmente se han adoptado diversas estrategias para mantener en funcionamiento software heredado. Una técnica ampliamente difundida es la aplicación de SOA [223, 196, 91]. Otra estrategia es exponer la funcionalidad del software heredado a través de envoltorios (wrappers) utilizando servicios web [42, 222, 141, 11, 102]. En tercer lugar se encuentra el enfoque de la utilización de sistemas multi-agentes como el utilizado en [138].

2.5. El Software Científico Heredado

Una afirmación interesante sobre el rol que cumple el software científico se encuentra dada por Judith Segal en [163], en el cual se afirma que el software científico heredado actúa como medio por el cual una vieja generación de científicos le puede pasar el conocimiento encapsulado durante años a una nueva generación de científicos. En esta afirmación se ve reflejado el carácter hereditario del conocimiento y del software. Cabe destacar que se ha encontrado en la bibliografía una serie de afirmaciones que indican que los métodos de producción de software científico actuales divergen significativamente de los métodos de producción de software de la industria. En sus inicios, hacia la década del 40 todo el cómputo era realizado por aplicaciones construidas por científicos, pues muy difícilmente una persona común tuviera acceso a una computadora en esos años. Por ello el concepto de computación en esa época era sinónimo de computación científica [121]. Desde los inicios de las Ciencias de la Computación, término utilizado por primera vez en 1959 [81], ha adquirido mucha influencia en la vida de las personas. Desde entonces y en unas seis décadas las Ciencias de la Computación han evolucionado en un vasto número de subcampos incluyendo a la Ingeniería de Software [195]. En comparación la evolución de la computación científica parece haber tomado un camino diferente, mucho más lento. Además, no sólo parece que ésta ha quedado atrás sino que al parecer ha tomado una dirección totalmente distinta. Algunos autores se refieren a este fenómeno como al “desajuste” o al “abismo” entre la Ingeniería de Software y la Computación Científica [121, 21, 71].

Las prácticas de construcción de software actuales utilizadas por los científicos

computacionales, muy a menudo tienen poca semejanza con los promovidos por los ingenieros de software [21, 194, 44, 106, 71, 121]. Existen varios posibles motivos que pueden llegar a explicar este hecho, podría ser que comúnmente el usuario final y el programador residan en la misma persona [194, 122, 21, 216, 44, 106, 121, 193]. Incluso cuando el poder del hardware y de cómputo están creciendo a una velocidad muy rápida, la Ciencia Computacional parece estar rezagada en el tiempo [216, 71]. Entre algunas de las razones del arriba mencionado desajuste en el desarrollo del software científico, se puede empezar a destacar la complejidad del dominio [194, 193]. Como segundo factor, la poca importancia que los científicos le otorgan al desarrollo de software y a las habilidades necesarias para construirlo [21, 194, 44]. El tercer problema es la escasa capacitación en las técnicas de la Ingeniería de Software y la calidad horizontal de la enseñanza, donde los conocimientos son pasados de un científico a otro ambos dos con el mismo nivel de competencias de programación [216, 21].

También es destacable que los científicos son reticentes a la utilización de herramientas modernas de desarrollo como lo son los Entornos de Desarrollo Integrados (IDEs) [216, 21, 44, 45].

A pesar de la falta de correspondencia existente entre la Ingeniería de Software y la Computación Científica los programas creados por los científicos son exitosos [192, 177, 176].

2.6. El Software Científico, La Ingeniería de Software, HPC y Fortran

Desde siempre uno de los problemas y cuello de botella que los científicos han tenido al ejecutar su software ha sido el tiempo de cálculo o tiempo de cómputo. Uno de los ejemplos más relevantes se encuentra en la simulación del clima realizada en 1949 por Jule Charney que confeccionaba una predicción del clima de 24 horas y tardaba 24 horas en computar los cálculos de dicha simulación [74, 215, 143]. Un ejemplo más actual es el del modelo climático WRF (The Weather Research & Forecasting Model) el cual dispone de un benchmark para medir

su performance, "WRF (benchmark) input files for 2001102400 case: 6 h interval, 48 h total, 425x300x34 grid", ver Figura 2.5.

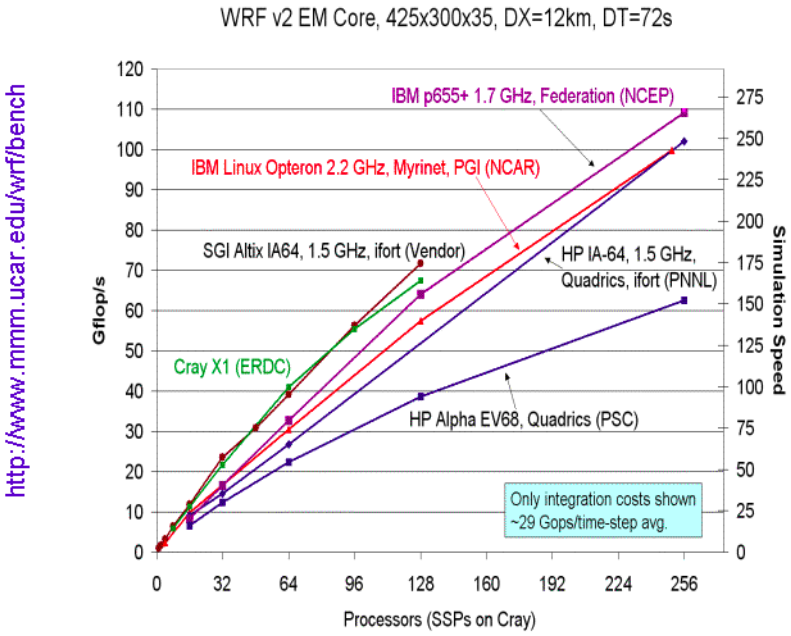


Figura 2.5: WRF (Benchmark)

Desde este punto de vista la evolución del hardware está íntimamente ligada con parte de los requerimientos del software científico, dado que este último necesita del poder de cómputo del primero. Desde hace muchos años los fabricantes de hardware han sido capaces de mejorar la performance de los microprocesadores. Por ejemplo, durante los años 90, haciendo que los microprocesadores dupliquen su poder en un factor de 2 cada dos años o más precisamente cada 18 meses, como lo predice la ley de Moore [161]. Además de la duplicación de la cantidad de transistores por unidad de área en los microprocesadores, este aumento de poder lo permitió el aumento lineal del reloj del microchip en la misma unidad de área [5]. En el año 2000, la velocidad en el incremento de poder en los microprocesadores se vio disminuida sustancialmente afectando así la evolución arquitectural de los chips. Mientras los procesadores se hacían más pequeños y la velocidad de los mismos aumentaba, la performance de los procesadores no podía superar un umbral del 12.5 %. En ese mismo año, IBM lanza al mercado el procesador POWER4, que

es considerado el primer microprocesador multi-core [94]. Este microprocesador poseía unos 170 millones de transistores en toda su superficie, estaba compuesto por dos núcleos corriendo a una velocidad 1.33 Ghz. Además el chip contenía: "1.44 MB of shared L2 cache memory plus the directory for a 32MB off-chip cache, a 500-MHz interconnection fabric, high-bandwidth buses and I/O designed to allow building an eight-way system on a single multi-chip module, and the logic needed to support large SMPs " en 180nm [199]. La era multi-core había visto la luz del día. Sin embargo, este paso adelante en la arquitectura de los procesadores tenía sus propio cuello de botella: la Pared de la Potencia (The Power Wall). El consumo de energía fue el problema que comenzó a preocupar a la industria de los microprocesadores, por ejemplo el Pentium 4 M alcanzaba unos 100 Watts por cm2; este hecho hizo que Intel interrumpiera la producción de estos procesadores. Si bien la Ley de Moore estaba en su apogeo, ciertos problemas saltaron a la luz:

- La realidad llevó el nivel de calor producido por los microprocesadores a un nivel cercano al de un reactor nuclear.
- Los arquitectos de procesadores se hallaban con el hecho que mientras los transistores eran cada vez más rápidos, el cableado interno era más lento, esto es conocido como Wire Delay.
- La arquitectura multi-core se encontró con la ley de Amdahl [5], que afirma que un programa que utiliza múltiples procesadores está limitado por la fracción secuencial del mismo.
- El espacio para alojar a los núcleos es limitado y además debe alojar FPU (Floating Point Unit), LSU (Load-store unit), IDU (Instruction Decode), IFU (Instruction Fetch Unit) y el subsistema de memoria (memorias cache L1, L2, L3).
- El factor de contracción o shrink factor: en 2014 la arquitectura IA-64 de Intel llegó a una litografía de 22nm, conocida como Haswell. Existen planes de llegar en poco tiempo a Broadwell de 14 nm de litografía y a Skylake de 10 nm.

Todos los factores anteriores son los problemas que ha encontrado la arquitectura multi-core en sus inicios, y algunos de éstos siguen siendo relevantes en la actualidad. Pero sin ningún lugar a dudas uno de los factores más importantes que no está en la lista anterior, es la existencia de una enorme (por no decir gigantesca) cantidad de software que no fue ni está concebido para utilizar las capacidades que ofrece un procesador multi-core. Incluso cuando los compiladores han evolucionado junto a los procesadores, éstos no son capaces de transformar un programa secuencial en uno con la misma funcionalidad pero de ejecución paralelo. Si bien son capaces de introducir mejoras a nivel de paralelismo de instrucción, aplicar transformaciones automáticas que aprovechen las características multi-core en los lenguajes de alto nivel, todavía es un desafío.

Desde la perspectiva de la Ingeniería de Software las aplicaciones de Cómputo Científico poseen ciertas características que las hacen interesantes como caso de estudio:

- Se enfocan en un problema específico y particular.
- Muchas de ellas han estado corriendo en ambientes de producción durante muchos años.
- Son un ejemplo de la vida real.
- Son aplicaciones de cómputo intensivo.
- Han sido evaluadas satisfactoriamente desde la perspectiva del problema que ellas resuelven.
- Están basadas en modelos matemáticos y numéricos que describen un fenómeno físico comprobable.

Como se ha dicho en el capítulo anterior muchos autores afirman que Fortran es el lenguaje de programación más utilizado en la construcción y desarrollo de programas de cómputo científico [34, 214, 66, 120, 28, 166, 162, 158, 44, 21, 189, 58].Por ello ha sido elegido como caso de estudio en este trabajo de investigación.

2.7. Trabajos Relacionados

La idea de utilizar transformaciones automáticas en el código fuente existente para aplicar cambios al mismo, es estudiada desde hace años por el Mantenimiento de Software. Sería muy ingenuo pensar que una nueva versión de una aplicación existente comience totalmente desde cero, efectivamente entonces podemos pensar que "el desarrollo de software es transfomar programas" [119]. Muchos procesos de desarrollo están concebidos para realizar software desde cero, como por ejemplo el modelo de desarrollo espiral, el modelo de desarrollo en cascada, el Rational Unified Process, entre otros. Bajo este concepto lo propuesto es asumir que habitualmente un sistema de software se crea desde la nada, es decir, que esto sería lo normal. Por otro lado, la aplicación de cambios a un sistema existente se denomina Mantenimiento de Software. Según IEEE el mantenimiento es "La modificación de un producto de software después de su entrega al cliente o usuario para corregir defectos, para mejorar el rendimiento u otras propiedades deseables, o para adaptarlo a un cambio de entorno" [112]. Si tenemos en cuenta esta definición parecería ser que el mantenimiento o la reingeniería de software son un caso especial y son tratados como excepción a la norma dentro del proceso de desarrollo [119]. Por el contrario está ampliamente estudiado que la etapa de mantenimiento del software consume la mayor parte de los recursos de un proyecto. Pudiendo así considerarse que desarrollar software de cero es el caso excepcional y que lo normal es transformar software preexistente [119].

2.7.1. Mantenimiento de Software

A partir de los años 70 se comenzó a vislumbrar la importancia de esta etapa dentro del proceso de desarrollo de software, incluso se dividió al mantenimiento en tres distintas dimensiones [197]. Ya hacia 1972 un autor describió al mantenimiento como un Iceberg, de aquí nace el concepto del Iceberg del Mantenimiento por el cual se supone que hay muchos factores dentro del proceso de mantenimiento que están ocultos [43]. En 1976, E. Burton Swanson describe las tres dimensiones del mantenimiento [197]:

1. Mantenimiento Adaptativo: "*Maintenance performed in response to changes in data and processing environments may be termed adaptive maintenance. The timely anticipation of environmental change is necessary to insure effective performance of this type of maintenance*".

2. Mantenimiento Correctivo: "*Maintenance performed in response to failures of the above types may be termed corrective maintenance. Especially where processing failures are concerned, a diagnosis of the causes of failure constitutes a significant portion of the task for this type of maintenance activity*".

3. Mantenimiento Perfectivo: "*Maintenance performed to eliminate processing inefficiencies, enhance performance, or improve maintainability may be termed perfective maintenance. Its aim is to make the program a more perfect design implementation. It is undertaken when "justified,ï.e., when the improvements to be achieved outweigh the costs of making those improvements*".

Una cuarta dimensión es introducida en trabajos como en [117] llamado Mantenimiento Preventivo: "*the modification of a software product after delivery to detect and correct latent faults in the software product before they become operational faults*" [117].

Si bien al parecer existen definiciones formales dentro de la bibliografía sobre qué es el mantenimiento y cuáles son sus dimensiones, existen autores que utilizan distintas definiciones para los mismos conceptos, eso puede verse en [46] en el cual se muestra cómo estos términos son utilizados según distintos autores.

Desde hace años los investigadores y la industria han encaminado sus esfuerzos en encontrar y estudiar la forma de mejorar el proceso de desarrollo de software y su mantenimiento [135]. A partir de algunos de estos estudios se ha caracterizado al software con cuatro propiedades esenciales del software: complejidad, conformidad, cambio, intangibilidad [38], así como también leyes que describen la evolución del software a través del tiempo [133, 134, 135]. Estas propiedades afectan directamente al proceso de mantenimiento del software. Este proceso o etapa en el desarrollo de software ha ido ganando relevancia a través de los años.

Ya hacia 1969, se calculaba que el porcentaje del esfuerzo requerido en la etapa de mantenimiento, dentro del marco del proceso de desarrollo de software, era de entre el 40 y el 60 %. En el articulo Lienzt et al. [140] de 1978 ya se hace referencia a otro artículo publicado 10 años antes sobre mantenimiento de software por Riggs [184], en el cual varios autores hacían referencia a estos valores. En 1979 el costo relativo de la etapa de mantenimiento según [220] rondaba el 67 %, otros trabajos escritos en esa década ya señalaban que este valor podría rondar el 70 % [30]. Hacia 1981 el costo relativo del mantenimiento ascendía a más del 50 % en más de 487 organizaciones, posicionándose entre el 50 y el 75 % según [147], manteniéndose en esos valores según [179, 110] en los años 1988 y 1990 respectivamente. Se podría pensar que a partir de la década de los 90 esa tendencia iba gradualmente a decrecer. Sorprendentemente, esta tendencia fue creciendo a más del 75 % del costo relativo [73] hasta alcanzar más del 90 % del costo total relativo del proyecto [159, 100, 78, 191, 210, 59]. Si bien todos estos valores han sido calculados por diferentes métodos y técnicas lo que claramente están destacando es cuán costoso, complejo e importante es el proceso de mantenimiento dentro del proceso de desarrollo de software. Para concluir en un trabajo realizado por H. Bohen [29] fue publicada una serie de puntos sobre la detección de fallas en el proceso de desarrollo de software:

- Encontrar y arreglar un problema de software hallado después de haber sido entregado el producto es 100 veces más costoso que encontrar y arreglar un problema de software en la etapa de elicitación de requerimiento y/o en la etapa de diseño.

- Cerca del 80 % de los defectos se hallan en el 20 % de los módulos. Cerca del 50 % de los módulos están libres de errores.

- El 90 % del tiempo de inactividad de un sistema es debido, como mucho, al 10 % de los defectos.

Los Números del Mantenimiento de Software

Uno de los aspectos fundamentales del presente trabajo se basa en los números que se desprenden del proceso de mantenimiento del software. No existe un trabajo científico que estime con precisión la cantidad de líneas de código fuente existentes hasta la actualidad. Algunas estimaciones son :

- G. Booch estima que entre 1945 y 2005 se produjeron acumulativamente 1.000.000.000.000 de líneas de código fuente modificadas o nuevas con un incremento anual de 35.000.000 de líneas de código. [32].
- Kontogiannis et al. estimaron que al inicio de la década del 2000 en el mundo existían 800.000.000.000 líneas de código escritas [124].
- Lammel también estimaba, en el 2001, que existían alrededor de 1.000.000. 000.000 de líneas de código fuente escritas, de las cuales el 30 % de COBOL (225 millones de LOC), el 20 % en C/C++ (180 millones de LOC) el 10 % en Assembler (140 a 220 millones de LOC) y de otros lenguajes menos comunes el 40 % (280 millones LOC) [128].

El número de líneas de código fuente que estaban en proceso de mantenimiento fue estimado en 200.000.000.000 en 2001 por [78]. Para tener una noción del crecimiento de la producción de software y por ende del mantenimiento del mismo en un solo lenguaje de programación, a continuación se muestra el incremento en LOC del software construido en COBOL entre 2000-2008 según varios trabajos realizados sobre el tema:

- 100.000 millones (2000) [57].
- 225.000 millones (2001) [128].
- 200.000 millones (2005) [18].
- 180.000 millones (2006) [212].
- 200.000 millones (2008) [3].

El proceso de mantenimiento de software no puede pasar desapercibido, además cabe destacar que es muy difícil obtener datos actualizados sobre la situación del software que se encuentra en mantenimiento en nuestros días.

2.7.2. Reestructuración de Código Fuente

A partir de la complejidad y costo que conlleva la etapa de mantenimiento de software, los investigadores han puesto foco a lo largo de los años en desarrollar técnicas y herramientas para dicha etapa. En este trabajo se ha rastreado hacia atrás el posible primer uso de "restructuring" (o reestructuración en español) como concepto en el artículo de GUY de BALBINE de 1976 llamado "Better manpower utilization using automatic restructuring" [60]. Básicamente el concepto proviene de los trabajos realizados en la década de los 70 para transformar programas no estructurados (unstructured), escritos utilizando la instrucción GO TO, en programas estructurados basados en el paradigma de la programación estructurada [31, 67, 206].

Según Arnold la reestructuración es: "Software restructuring is the modification of software to make the software easier to undertstand and to change, or less suceptible to error when future changes are made" [10].

Una definición más general y abarcativa de lo que es la reestructuración de código fuente, es la dada por E. J. Chikofsky en su trabajo "Reverse engineering and Design Recovery : A Taxonomy":"Restructuring is the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics)" [49].

En la definición proporcionada por Chikofsky queda claro que la reestructuración de código fuente es una transformación que se le aplica al código existente para obtener otra representación. En la mayoría de los casos mejor, manteniendo el nivel de abstracción y además, manteniendo el comportamiento externo del mismo. Según Chikofsky el térmimo reestructuración abarca un amplio espectro de transformaciones que no solamente se limitan a código fuente sino que también

pueden ser utilizadas para transformar modelos de datos, estructuras de requerimiento, planos de diseño, etc. Por ejemplo, la normalización es para el autor, un ejemplo de transformación de datos a datos que implica una mejora en el modelo lógico de datos en el proceso de diseño de base de datos [49].

La reestructuración nace como un proceso necesario para implementar el mantenimiento de software debido a las características esenciales del mismo (complejidad, conformidad, cambio, intangibilidad) [38] y con el fin de reducir los costos en el proceso de mantenimiento de éste. Asimismo, la inherente complejidad del software se ve incrementada aún más cuando al software ya existente se le deben realizar cambios de alguna naturaleza (correctivos, adaptativos, perfectivos o preventivos), creándose así un ciclo en el cual el software se torna cada vez más y más complejo de manejar. Teniendo en cuenta lo anteriormente dicho, la reestructuración tiene como objetivo, desde cierto punto de vista, reducir la complejidad del software mediante la aplicación incremental de mejoras en su estructura interna [10, 101]. Existe un caso particular cuando la bibliografía se refiere a sistemas construidos bajo el paradigma de la programación orientada a objetos en la cual al proceso de reestructuración se lo denomina refactorización. La refactorización de código fuente es básicamente la variante orientada a objetos de la reestructuración [156] cuya definición según Martin Fowler es: “the process of changing a [object-oriented] software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure” [89]. Un problema asociado a la reestructuración de código fuente, es la aplicación de dichas reestructuraciones de forma manual. Efectivamente este proceso aplicado manualmente tiende a ser costoso y propenso a errores [101], por ello es fundamental la aplicación de herramientas automatizadas para llevar a cabo estas transformaciones de código fuente.

Reestructuración de Código Fuente Fortran

Una de las características más importantes de Fortran es su longevidad, esta longevidad hace que los primeros estudios de reestructuración de código fuente se

remonten a fines de los años 60 y principios de los años 70. En estos primeros estudios se trataba de estructurar o reestructurar código fuente llamado "spaghetti" específicamente para Fortran [109, 60, 16, 15, 98, 75]. Existen además en los últimos años varios trabajos orientados a reestructurar automáticamente utilizando herramientas automáticas de transformación de código fuente Fortran secuencial con el objetivo de paralelizarlo [131, 204, 178, 65, 65, 20, 201, 203].

Otros trabajos han realizado reestructuración de código fuente Fortran [169] [174] [168] [173] implementando dichas transformaciones como refactorizaciones de código fuente. En [149] se ha descrito un catálogo de transformaciones de código fuente con más de 30 transformaciones, muchas de las cuales han sido implementadas e integradas a Eclipse bajo la forma de refactorizaciones.

2.7.3. Herramientas de Reestructuración y Mantenimiento para Fortran

A continuación se presentará una lista detallada de las herramientas comerciales y gratuitas escritas para Fortran cuyos objetivos son el análisis de programas y la reestructuración de los mismos. Las herramientas analizadas se listan en forma resumida a continuación:

1. **Diagramf**
2. **Ftnchek**
3. **plusFORT**
4. **FOR_STRUCT**
5. **ForCheck**
6. **ConvertF90**
7. **Fortran Lint**
8. **NAGware tools**
9. **Visual StrongType for FORTRAN 77**

10. **FORTRAN 77 Flowcharting Utility**
11. **Floppy**
12. **f77chk Ver.1.2.2e**
13. **OmegaChart ver. 6.0**
14. **WinFPT**
15. **ForUtil**
16. **FCAT**
17. **FORESYS**
18. **Vast F77 to F90**
19. **The NAGWare tools**

Descripción Detallada de las Herramientas

I **Diagramf**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Interfaz**: Consola
- **Descripción**: Esta aplicación fue desarrollada en 1995 por Mitchell R Grunes (grunes@nrlvax.nrl.navy.mil). La idea de este desarrollo es la de marcar los inicios y fines de las diversas estructuras de control que posee Fortran. Estructura IF-ELSE-ELSEIF-ENDIF, DO-ENDDO y case. Además de marcar el inicio y fin de las rutinas, definiciones de tipos de datos, módulos e interfases. También reconoce los GO TO, RETURN, CYCLE, EXIT, y STOP con un *. Este programa está escrito enteramente en FORTRAN 77, el resultado obtenido de la ejecución de esta aplicación sobre un programa Fortran se ve en el siguiente ejemplo:

```
c      +---------------- subroutine a(x)              1
c      |+--------------- do i=1,5                    2
c      ||+-------------- if(i/2*2.eq.i)then          3
c      |||                  x=x*i                    4
c      ||+-------------- else                        5
c      |||                  x=x/i                    6
c      ||+-------------- endif                       7
c      |+--------------- enddo                       8
c      +---------------- end                         9
```

http://gd.tuwien.ac.at/languages/fortran/freemarket/diagram.html

II **Ftnchek**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Interfaz**: Consola
- **Descripción**: Este programa fue diseñado por el Dr. Robert Moniot, profesor en la Universidad de Fordham. La idea principal bajo el diseño de la aplicación no es la de detectar errores sintácticos en los programas Fortran, sino más bien focalizarse en la detección de errores semánticos dentro de los programas. Estos errores semánticos son partes de código fuente Fortran que no causan errores de sintaxis, pero sí pueden causar la incorrecta utilización de los recursos y el funcionamiento incorrecto del programa. Ejemplos de este tipo de errores se encuentran en variables declaradas y no utilizadas, variables que no han sido inicializadas, etc. Actualmente según está publicado en su sitio web la última versión del producto es la 3.3. Cabe destacar que es una aplicación de consola y es de código abierto.

http://www.dsm.fordham.edu/ftnchek/

III **plusFORT**:

- **Tipo de Software**: Análisis estático, dinámico y Reestructuración

- **Tipo de Interfaz**: Consola
- **Descripción**: plusFort es un kit de herramientas de análisis estático y de reestructuración de código fuente escrito en Fortran. Desarrollado por la empresa Polyhedron Software es un producto pago que permite hacer: reestructuración, reformateo, análisis estático y análisis dinámico de programas escritos en Fortran. Entre algunas de estas herramientas se encuentran: SPAG, AUTOMAKE, GXCHK, entre otros. Existen versiones disponibles para plataformas Mac, Linux y Windows.

http://www.polyhedron.com/products/fortran-tools/plusfort-with-spag/plusfort-versión-6.html

IV **FOR_STRUCT**:

- **Tipo de Software**: Reestructuración
- **Tipo de Intefaz**: Consola
- **Descripción**: Es una herramienta que permite realizar reestructuraciones de instrucciones IF-GOTO, ASSIGNED GOTO, COMPUTED GOTO y ARIHMETIC IF en estructuras de control modernas. Es capaz de analizar distintas versiones de Fortran: FORTRAN IV, FORTRAN 66, FORTRAN 77 y Fortran 90. Es un producto pago que se utiliza bajo licenciamiento de la empresa Cobalt-Blue.

http://www.cobalt-blue.com/fs/fsmain.htm

V **ForCheck**:

- **Tipo de Software**: Análisis Estático y Documentación
- **Tipo de Intefaz**: IDE Propio y consola
- **Descripción**: Esta herramienta es una de las más antiguas que existen en el mercado, según sus creadores. Es básicamente una herramienta de análisis estático de código fuente que genera informes de múltiples

posibles mejoras y usos incorrectos del lenguaje de programación. Identifica los errores de codificación y código muerto; detecta variables no utilizadas, sin referencias o no definidas; detecta punteros y variables no asignados; verifica las referencias y argumentos de procedimiento; identifica sintaxis Fortran obsoleta, código no portable, entre otros. Además realiza verificación de referencias y listas de argumentos de procedimiento; verificación y detección de uso inconsistente de COMMON BLOCKS; detecta COMMON BLOCK sin usar, sin definir y objetos no referenciados, entre otras características. Desarrollado para plataforma Windows y Linux, es un producto pago con una versión de prueba.

http://www.forcheck.nl/

VI **ConvertF90**:

- **Tipo de Software**: Reestructuración
- **Tipo de Intefaz**: Consola
- **Descripción**: Esta aplicación fue creada por Michael Metcalf (metcalf@cernvm.cern.ch), su código fuente se encuentra distribuido para ser compilado por cualquier compilador Fortran. Ésta es una herramienta que transforma código fuente de programas escritos en FORTRAN 77 a código fuente Fortran 90. A medida que el código fuente es transformado, también indenta los bloques de las estructuras de control IF y DO, además reemplaza el CONTINUE por END DO, agrega los nombres de las rutinas en los END. Puede ser utilizado en cualquier plataforma Mac, Linux o Windows dependiendo del compilador. Es un programa de código abierto.

https://wwwasdoc.web.cern.ch/wwwasdoc/WWW/f90/convert.f90

VII **Fortran Lint**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Interfaz**: Interfaz Gráfica

- **Descripción**: Esta herramienta según sus creadores identifica posibles problemas en el código Fortran: portabilidad, sintaxis, uso de variables, inconsistencas en las declaraciones de COMMON BLOCKS, variables y rutinas no utilizadas y construcciones obsoletas de los lenguajes FORTRAN 77 y Fortran 90. Además realiza chequeos de inconsistencias en las declaraciones de COMMON BLOCKS, chequea la portabilidad del código fuente, chequea el uso Tipos a través de diferentes subprogramas / unidades de programa, código muerto, problemas de uso de variables, entre otros. Genera árboles de llamadas, produce una tabla de símbolos de referencia cruzada con la información detallada sobre todos los símbolos globales y produce informes HTML. Es una de las pocas herramientas que tiene opciones para detectar posible paralelización utizando un esquema de memoria compartida, en este caso OpenMP.

`http://stellar.cleanscape.net/products/fortranlint/`

VIII **NAGware tools**:

- **Tipo de Software**: Análisis Estático y Reestructuración
- **Tipo de Interfaz**: Consola e Interfaz Grafica
- **Descripción**: Este conjunto de herramientas permite realizar transformaciones de código fuente que van desde reformatear código fuente hasta la realización de análisis de dependencias y la construcción de árbol de llamadas. También posibilita la aplicación de transformaciones de código fuente para reemplazar estructuras obsoletas del lenguaje por otras más modernas.

`http://www.nag.co.uk/nagware.asp`

IX **Visual StrongType for FORTRAN 77** :

- **Tipo de Software**: Análisis Estático
- **Tipo de Interfaz**: Gráfica

- **Descripción**: Esta herramienta está desarrollada en un entorno gráfico y discontinuada por sus creadores, analizaba la consistencia algebraica en el programa. Aún se puede obtener una versión de prueba de 21 días, aunque en el sitio web de sus creadores indica que está discontinuada.

http://www.visualstrongtype.com/download.html

X **FORTRAN 77 Flowcharting Utility**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Intefaz**: Consola e Interfaz Gráfica
- **Descripción**: Este producto de código abierto permite analizar código FORTRAN 77 para construir posteriormente un diagrama de flujo del programa en cuestión. Fue construido por Vince Weaver <vincdeater.net> y utiliza software adicional para visualizar el grafo provisto por AT&T GraphViz utilities, ver Figura 2.6.

http://www.deater.net/weave/vmwprod/f77_diagram/

XI **Floppy**:

- **Tipo de Software**: Análisis Estático, Reestructuración, Documentación
- **Tipo de Intefaz**: Consola
- **Descripción**: Esta herramienta realiza varios análisis sobre el código fuente Fortran tomando como entrada un archivo escrito en FORTRAN 77. Está basado en FLOP (FORTRAN Language Orientated Parser) y entre sus características se encuentra: análisis de código, pretty printing, chequeos de consistencia de uso de parámetros y COMMON BLOCKS. Genera una versión navegable del código en HTML. Es de código abierto y está escrita en Fortran.

http://www.netlib.org/floppy/

XII **f77chk Ver.1.2.2e**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Intefaz**: Consola
- **Descripción**: Esta utilidad fue creada por Tsuguhiro TAMARIBUCHI (tamari@spdg1.sci.shizuoka.ac.jp) y consiste en un programa escrito en Perl, que chequea la consistencia de los argumentos en cada llamada de una subrutina. También realiza chequeos sintácticos de las instrucciones IF, DO y CALL.

http://spdg1.sci.shizuoka.ac.jp/f77chk/indexe.html

XIII **OmegaChart ver. 6.0** :

- **Tipo de Software**: Creación de Diagramas de Flujo
- **Tipo de Interfaz**: Gráfica
- **Descripción**: Esta herramienta crea diagramas de flujo a partir de código fuente escrito en FORTRAN 77 o Fortran 90. Posee dos formatos de salida: uno en formato Excel y otro en formato HTML. Esta herramienta puede ser utilizada para otros lenguajes de programación.

http://home.comcast.net/~lchen223621/

XIV **WinFPT**:

- **Tipo de Software**: Análisis Estático, Reestructuración y Documentación
- **Tipo de Interfaz**: Interfaz Gráfica y Consola
- **Descripción**: Esta herramienta permite realizar tareas de mantenimiento desde varias perspectivas. La primera mediante la generación de métricas y reportes (árbol de llamadas, tabla de símbolos entre otros). La segunda mediante la detección de problemas en el código fuente como por ejemplo código muerto, variables muertas, errores en expresiones, etc. La tercera perspectiva mediante la aplicación de restructuraciones

de código fuente como ser: renombrar variables, agregar IMPLICIT NONE, reemplazar COMMON BLOCKS, etc. La cuarta perspectiva es la de la posibilidad de formatear el código fuente haciendo Pretty Printing de los programas. Y por último realiza análisis dinámicos sobre cobertura de test.

http://www.simcon.uk.com/

XV **ForUtil**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Interfaz**: Consola
- **Descripción**: Este es un conjunto de herramientas que permite obtener información de programas escritos en FORTRAN77. La primera de estas herramientas llamada fflow genera un diagrama de flujo del programa. La segunda ffscan se encarga de chequear la consistencia entre los argumentos de las subrrutinas y las llamadas. Por último un conjunto de tres programas que realizan análisis de COMMON BLOCK llamados get_common, list_commons y scan_commons. Este set de herramientas creadas por Koen D'Hondt se encuentra disponible sólo para Linux.

http://dev.man-online.org/package/main/forutil/

XVI **FCAT**:

- **Tipo de Software**: Cobertura de Código
- **Tipo de Interfaz**: Consola
- **Descripción**: Esta herramienta permite determinar el grado de cobertura del código fuente. Si bien este concepto se utiliza en testing, aquí se refiere a determinar si una instrucción es ejecutada o no. Según sus creadores Fcat permite determinar puntos fríos o puntos calientes en el código fuente. Esta herramienta toma como entrada código fuente escrito en Fortran y produce un archivo de salida con el mismo código

fuente con el agregado de marcas de no ejecución en este caso *> y en las líneas ejecutadas se marcan la cantidad de veces que ésta se ejecuta. Está escrito en Perl y puede ser descargado de la web gratuitamente para Windows o Linux.

http://yifanhu.net/SOFTWARE/FCAT/

XVII **FORESYS**:

- **Tipo de Software**: Análisis Estático
- **Tipo de Interfaz**: Interfaz Gráfica
- **Descripción**: Esta herramienta permite realizar análisis interprocedurales, verificaciones de codificación, posee un módulo de paralelización y de generación de métricas de calidad. Acepta distintas extensiones de Fortran 90 y FORTRAN 77.

http://www.post.ecs.soton.ac.uk/foresys.html

XVIII **Vast F77 to F90** :

- **Tipo de Software**: Análisis Estático y Reestructuración
- **Tipo de Interfaz**: No especifica
- **Descripción**: Esta herramienta es un traductor de código fuente escrito en FORTRAN 77 a Fortran 90. Además de ser un formateador de código fuente realiza cambios en ciertos puntos del programa como eliminar características obsoletas, eliminar GOTOs y reducir la cantidad de etiquetas, crear módulos para las áreas de COMMON BLOCKS, entre otras. Una de las características llamativas del producto es que su licenciamiento es medido por la cantidad de líneas de código fuente que son restructuradas.

http://www.crescentbaysoftware.com/

XIX **The NAGWare tools**:

- **Tipo de Software**: Análisis Estático y Reestructuración
- **Tipo de Interfaz**: Consola
- **Descripción**: Es un conjunto de herramientas. Éstas permiten realizar distintos tipos de reestructuraciones. También permiten generar distintos tipos de información extraídas del análisis estático de código fuente. Entre algunas de las características de estas herramientas están: la generación del árbol estático de llamadas, generación automática del Makefile, análisis de dependencias, reformateo de código fuente, reestructuración de construcciones de código fuente obsoletas del lenguaje.

http://www.crescentbaysoftware.com/

2.7.4. Resumen

La problemática del mantenimiento del software en sí mismo se va haciendo más compleja a medida que el tiempo transcurre. El software científico por tener características especiales y diferentes a las del software comercial, hace que el mantenimiento y la modernización del mismo sea un proceso muy complejo. A su vez, la aplicación de cambios a este tipo de software también resulta una ardua tarea para ser llevada a cabo, y más aún manualmente. Las herramientas que se encuentran disponibles no parecen estar integradas a un entorno de programación. Este hecho hace que la tarea de mantenimiento se deba realizar utilizando múltiples herramientas distribuidas en una misma computadora, con lo cual además de resultar incómodo a la hora de realizar los análisis y aplicar los cambios, es altamente propenso a la producción de errores y de largos tiempos de ejecución ya que el programador debe ir cambiando de herramienta según lo necesite, lo que implica pérdida de tiempo y concentración.

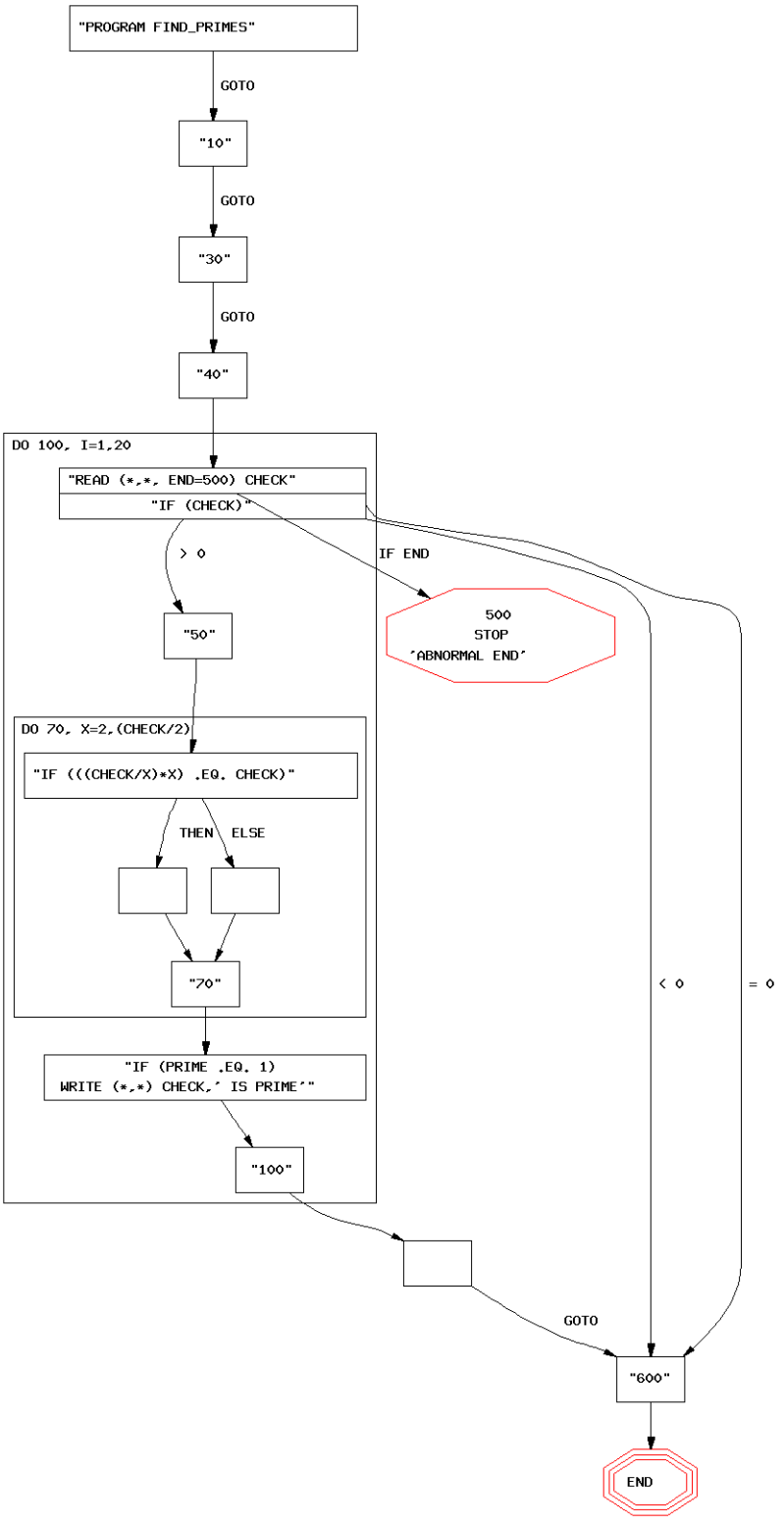


Figura 2.6: Ejemplo del Análisis Realizado por FORTRAN 77 Flowcharting Utility

Capítulo 3

Un Proceso de Desarrollo/Mantenimiento Dirigido por el Cambio

Una de las características esenciales del software es que éste está continuamente sometido a las presiones por el cambio. Existen muchos procesos de desarrollo pero ninguno que se base específicamente en esta característica esencial del software. En el capítulo se describe un proceso de desarrollo y mantenimiento que está dirigido por el cambio (Change-Driven).

3.1. Mantenimiento de los Sistemas Heredados Escritos en Fortran

En el capítulo anterior se ha revisado la historia de Fortran a través de los años, en la actualidad no existen estimaciones sobre la cantidad de líneas de código fuente escritas en este lenguaje. Por ello, se ha analizado repositorios de código abierto para encontrar, si existiera, alguna tendencia sobre el software escrito en Fortran en nuestros días. Se han estudiado dos conocidos repositorios de código fuente que son: GitHub y Open Hub, este último posee registro de 30.879.289.910

líneas de código almacenadas. Del análisis que se ha realizado a los proyectos que están almacenados en el repositorio Open Hub, se puede apreciar que la cantidad de líneas de código Fortran escritas en formato fijo casi duplica a la cantidad de líneas de código escritas en formato libre, ver Tabla 3.1.

	Fortran Free Format	Fortran FIXED FORMAT
Total Lines	27.397.164	43.810.157
Code	22.109.972	37.361.967
Comments	3.139.837 (12,4 %)	4.315.797 (10,4 %)
Proyectos	1.227	1.207
Desarrolladores	3.329	2.716
Commits	124.644	68.853

Cuadro 3.1: Open Hub: Datos de los Proyectos Escritos en Fortran

Aunque las tendencias que se ven en las Figuras 3.1, 3.2, 3.3, 3.4, 3.5 y 3.6 indican que la cantidad de actividad crece para los proyectos en formato libre, y decrece para los proyectos de formato fijo, la cantidad de código fuente en formato fijo prevalece sobre el formato libre. Existe en estos repositorios un 50 % más de líneas escritas en formato fijo que en formato libre.

En esta dirección, en base a la preponderancia del formato fijo, que además puede ser un indicador de que estos programas están escritos en versiones viejas de FORTRAN, se debe tener en cuenta que existen leyes que rigen la evolución natural de los programas y de los sistemas de información a través del tiempo a las que estos programas también están sujetos (las leyes de "Lehman") [134, 137, 135, 133, 136]. Además, como si esto no fuera suficiente, algunos autores como Brian Foote y Joseph Yoder proponen que el patrón arquitectural más utilizado en la producción de software es el que ellos denominan La Gran Bola de Barro (Big Ball Of Mud) y otros patrones relacionados que fueron descritos en [82]. Son por ende los patrones arquitecturales que en la práctica se encuentran aplicados al código fuente en mantenimiento, al que se encuentra en producción o al que está siendo construido. A continuación se describen detalladamente algunos de los propuestos en [82]:

- Gran Bola de Barro (Big Ball of Mud): Un ente en expansión descontrolada,

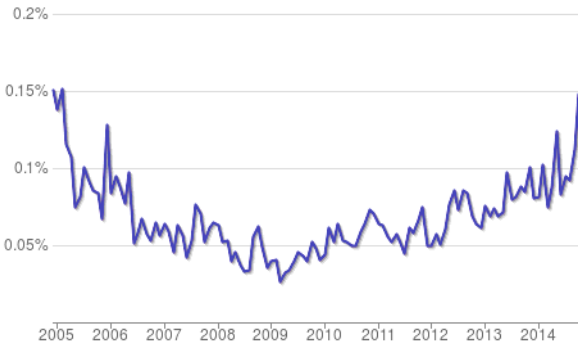


Figura 3.1: Cantidad de Commits Mensuales por Proyecto Fortran en Formato Libre en Open Hub

Figura 3.2: Cantidad de Gente que Contribuye Mensualmente en los Proyectos Para Fortran en Formato Libre en Open Hub

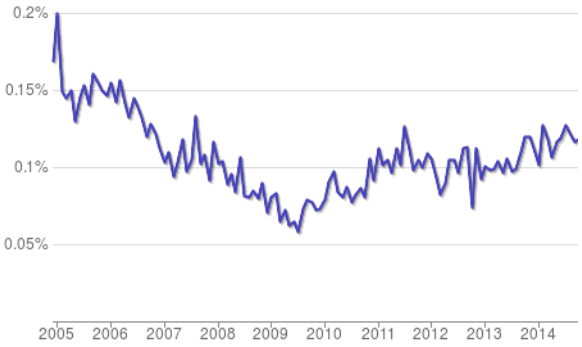


Figura 3.3: Cantidad de Proyectos Mensuales Para Fortran en Formato Libre en Open Hub

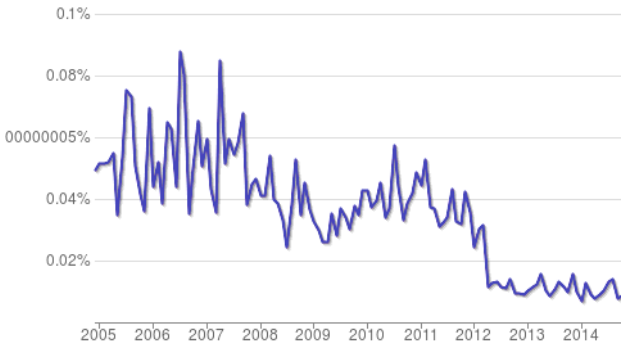


Figura 3.4: Cantidad de Commits Mensuales por Proyecto Fortran en Formato Fijo en Open Hub

Figura 3.5: Cantidad de Gente Que Contribuye Mensualmente en los Proyectos Para Fortran en Formato Fijo en Open Hub

Figura 3.6: Cantidad de Proyectos Mensuales Para Fortran en Formato Fijo en Open Hub

su estructura está atada con alambres y con código spaghetti. Se ven signos de crecimiento no regulado y de arreglos ad-hoc.

- Código Desechable (Throwaway Code): Es código escrito a las apuradas y de cualquier forma que sirve para salir del paso. Habitualmente no está documentado y a pesar de su origen provisorio se perpetua en el tiempo.

- Crecimiento Fragmentado (Piecemeal Growth): Es ineludible que un programa tenga que enfrentarse a cambios de requerimientos. Incluso hasta los diseños más cuidados tienen que enfrentarse al cambio e ir gradualmente creciendo sin control alguno.

- Que Siga Funcionando (Keep it Working): A menudo la solución del problema que ataca el software pasa a ser dependiente del software mismo y necesita de éste para poder lograrse. Por ejemplo en un pronóstico del clima las simulaciones realizadas por los modelos son vitales para poder pronosticar, sin esas simulaciones los pronósticos se vuelven muy complejos de realizar. Debido a esta dependencia hay que mantener el programa funcionando a cualquier costo.

- Barrer Debajo de la Alfombra (Sweep it Under the Rug): El código desordenado, intrincado, difícil de entender, rebuscado y con características spaghetti es muy complejo de mantener, por ende es preferible mantenerlo acotado y fuera de la vista.

- Reconstrucción (Reconstruction): El código fuente ha llegado a un punto en el cual es inmanejable e incomprensible, es preferible construirlo de cero a mantenerlo.

Estos problemas arquitecturales son causados por una gran diversidad de factores. En el caso de Fortran son su característica evolutiva, el tipo de programadores que lo utilizan, la cantidad de tiempo que algunos programas llevan corriendo en producción y la características de los mismos podrían conllevar a que el software caiga en la Gran Bola de Barro.

3.2. De lo Desconocido a lo Conocido

Uno de los factores más críticos en el mantenimiento de software heredado escrito en Fortran (ya sea éste científico o no) es poder comprender aquello con lo cual se está trabajando. Esta afirmación que parece trivial, puede transformarse en fundamental justamente cuando se está trabajando con código fuente escrito por terceras personas. Este proceso de comprensión se torna aun más complejo cuando cuando se está trabajando con programas escritos hace más de veinte o treinta años, pues incluso la forma de escribir el código puede resultar compleja de entender. Por ello es fundamental encontrar un proceso que parta de lo "desconocido", en este caso el código fuente heredado escrito por otros y nos deje en lo "conocido", código fuente comprensible, legible y actualizado. Es muy complejo poder extraer conocimiento rápidamente a partir de código fuente escrito por otra persona, éste puede ser código fuente enrevesado, críptico, entre otras cosas. Lo anterior puede verse en un fragmento de código fuente extremo, que proviene de un ejemplo de la vida real (extraído de un modelo de predicción climática) ver Figura 3.7. En dicho fragmento no es fácilmente diferenciable qué "**if**" actúa como variable y qué "**if**" actúa como estructura de control. De hecho, la aplicación de un cambio llevado a cabo mediante las técnicas de Búsqueda y Reemplazo (Search and Replace) podría agregar efectos colaterales no deseados al mismo.

Uno de los grandes desafíos que tienen que enfrentar los programadores es el problema de la adquisición de conocimiento sobre la porción del código fuente del programa sobre el cual están trabajando. Resulta fundamental poder reconocer el "qué" y el "cómo" de dicha parte del código.

3.3. Un Modelo de Proceso de Desarrollo de Software Orientado al Cambio

Como se ha discutido en capítulos anteriores es más probable que se desarrolle un nuevo producto de software a partir de uno ya existente que comenzar todo desde cero. De hecho, no tendría sentido la construcción de bibliotecas o compo-

```
    ip2=ip1
    twopi=twopic
    if =1
    if ( is .eq. 2 ) twopi=twopic
200 if ( ip2 .ge. ip4 )go to 480
    if = if +1
    if cur = ifact ( if )
    if ( ifcur .ne. 2 )  go to 120
    if ( ip2 .gt. ip4 )  go to 120
    if ( ifact (if +1) .ne. 2 ) go to 120
    if = if +1
120 ip3=ip2 + ifcur
    if ( if ) 120 ,200 ,480
```

Figura 3.7: Ejemplo Extremo de Código Fuente Fortran Extraído de un Modelo Climático

nentes que no sean reutilizados. Teniendo en cuenta además, que la maleabilidad y las continuas presiones para el cambio son dos características esenciales del software [38] y que la etapa de mantenimiento del software es la que más recursos del proyecto consume es necesario definir un proceso de desarrollo de software que esté Orientado al Cambio (Change-Driven). De hecho esta orientación, la de aceptar y abrazar el cambio, es la que utilizan las llamadas Metodologías Ágiles, muchas de éstas están basadas en el "Manifiesto por el Desarrollo Ágil de Software". El manifiesto proclama lo siguiente [24]:

> "Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar: -Individuos e interacciones sobre procesos y herramientas
> -Software funcionando sobre documentación extensiva
> -Colaboración con el cliente sobre negociación contractual
> -Respuesta ante el cambio sobre seguir un plan."

Esta visión sobre la construcción de software es considerada como una propuesta guiada por el cambio, en contraposición a la utilizada por las metodologías

clásicas que son guiadas por un plan (Plan-Driven). Una cuestión importante a tener en cuenta en esta visión es justamente que el cambio es una de las características esenciales del software. De esta forma se centra el proceso de desarrollo del proyecto en dos de las cuatro características esenciales del software [38]. Para ello este proceso debería estar guiado por Transformaciones o Cambios que deban ser aplicados al software. Se puede decir entonces que un proceso de desarrollo orientado al cambio es un Proceso de Desarrollo Guiado por Transformaciones ("Transformation Driven Develpment"). También podríamos definir a este proceso como un Proceso de Desarrollo Guiado por el Cambio ("Change Driven Development") pues cambiar es sinónimo de transformar. Si bien ambos conceptos son sinónimos, a lo largo de este trabajo se hará referencia a este proceso como:"Change Driven Development" o CDD.

3.3.1. El Proceso de Desarrollo Guiado por el Cambio

Un Proceso de Desarrollo de Software Guiado o Dirigido por el Cambio (PDGC o CDD), puede ser considerado como una metodología ágil para el mantenimiento y desarrollo de software científico, posiblemente extensible a cualquier tipo de software. Éste se caracteriza por:

1. Estar **Enfocado al Cambio**: dado que el cambio o las presiones por el cambio son unas de las características esenciales del software [38] el CDD posee como unidad mínima de trabajo al Cambio. Un cambio deberá pertenecer a una de estas cuatro caracterizaciones:

   a) Cambio Correctivo

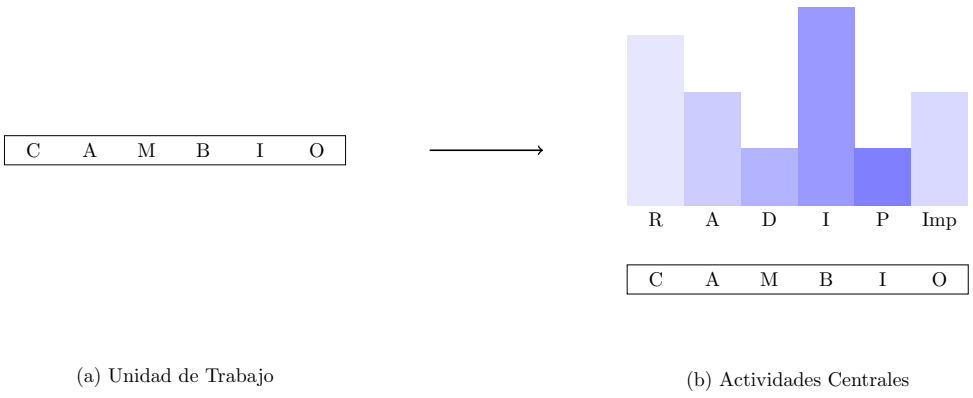
   b) Cambio Adaptativo

   c) Cambio Perfectivo

   d) Cambio Preventivo

   Un Cambio será, por consiguiente, la unidad de trabajo en la cual se podrán realizar las cinco Actividades Centrales (Core Activities) del desarrollo de software: Requerimientos, Análisis, Diseño, Implementación, Pruebas e

Implantación. Cada una de estas actividades tendrá un determinado porcentaje de ejecución dentro de un determinado tipo de cambio. En el caso de este trabajo se ha decidido que la suma de las actividades centrales totalizarán un valor de 100, que se dividirá según la participación de cada una de estas actividades dentro del cambio en sí mismo, ver Figura 3.8.



(a) Unidad de Trabajo

(b) Actividades Centrales

Figura 3.8: Descripción de un Cambio

2. Estar **Centrado en las Herramientas**: En este proceso de desarrollo son consideradas tan importantes como el cambio en sí mismo. Nadie en la actualidad puede concebir realizar cualquier actividad en el ámbito de la construcción de software sin la utilización de alguna herramienta de desarrollo. Desde este punto de vista son concebidas como el medio de aplicación del proceso en sí mismo en las cuales los programadores y demás integrantes del equipo de desarrollo deben sustentarse para la identificación, comprensión, transformación y verificación del software.

3. Ser un **Proceso Iterativo e Incremental**: El concepto de Proceso Iterativo e Incremental nace en la década de los 30 y 40 por los trabajos iniciales de Walter Shewhart en los laboratorios Bell, el cual introduce el concepto de realizar pequeños ciclos de "Plan-Do-Study-Act" [129]. Posteriormente retomado en los 40 por el padre de la Calidad W. Edwards Demings [129] el cual propone su famoso ciclo "Plan-Do-Check-Act". Curiosamente ambos son considerados los padres de los conceptos modernos de Calidad. El

> concepto a grosso modo de un Proceso Iterativo e Incremental reside en la construcción inicial del esqueleto de la solución del problema y a través de la ejecución de pequeños ciclos de trabajo (llamados iteraciones), en los cuales se realizarán ciertas tareas, para obtener la solución del problema al final de la aplicación del proceso completo [23].

A partir de las características que debe tener un proceso de desarrollo dirigido por el cambio, se propone una primera descripción de dicho proceso. Se ha de tener en cuenta que este enfoque considera cambio a cualquier variación en el estado de cualquier artefacto que intervenga en el proceso de desarrollo de software. Por ejemplo la obtención de una lista de cambios es un cambio en sí mismo: artefacto-> lista de cambios, estado inicial vacía, estado final llena. La visión de cambio o transformación no es únicamente aplicable al código fuente. El proceso puede partir de una lista de cambios ya existente o generar una iteración para obtener esa lista. Posteriormente, partiendo de un cambio a ser aplicado al sistema, el proceso se divide en cuatro fases. Estas fases son: Comprensión, Transformación, Verificación y Retroalimentación. A lo largo de estas cuatro fases se aplicará el Cambio al software. Para ello se ejecutarán en diferente intensidad las distintas actividades centrales del desarrollo de software (Requerimientos, Análisis, Diseño, Implementación, Pruebas e Implantación) que implique dicho cambio; y seguramente algunas otras actividades de soporte (Documentación, Managnement, entre otras muchas), al final del proceso el cambio es aplicado al sistema, ver Figura 3.9. Este ciclo podrá ejecutarse tantas veces como sea necesario, ver Figura 3.10.

Dentro de las cuatro fases del proceso de desarrollo guiado por el cambio se ha definido un flujo de trabajo específico. Se entiende por flujo de trabajo al estudio de los aspectos operacionales de una actividad en la cual se enfocan aspectos tales como: cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan, etc. En este caso para trabajar con aplicaciones científicas:

1. Establecer una versión inicial del código fuente

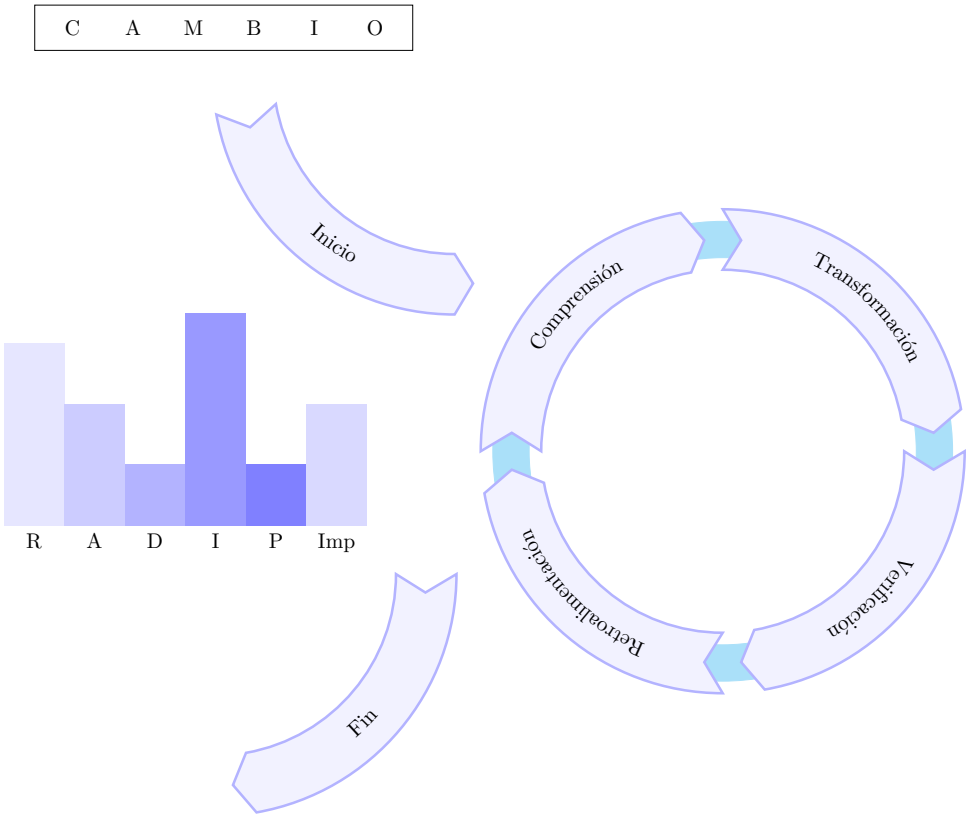


Figura 3.9: Un Ciclo Completo del Proceso

2. Transformar el código fuente
3. Verificar el código fuente obtenido
4. Validar los resultados numéricos
5. Aceptar/Rechazar el cambio en base a los resultados numéricos
6. Documentar

Cabe destacar que el proceso de desarrollo propuesto puede verse como la repetición de uno de estos ciclos tantas veces como se necesario, pues al final de cada iteración o ciclo la lista de cambios puede ser retroalimentada con nuevos cambios.

En las secciones siguientes describiremos las fases del proceso y el flujo de trabajo propuesto para cumplir todas las etapas.

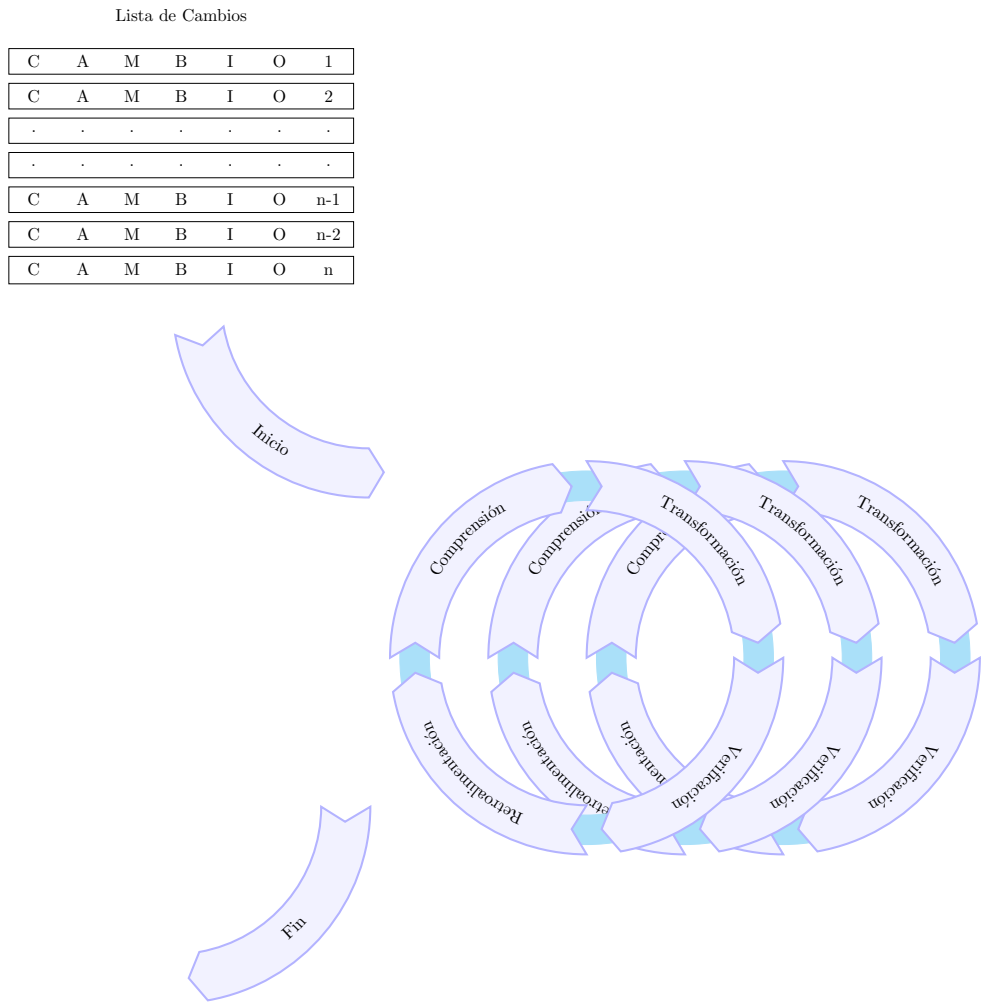


Figura 3.10: Fases del Proceso

Una característica fundamental de este proceso, como ya se ha comentado, son las herramientas a utilizar. Es de vital importancia que todas las herramientas que estén disponibles se encuentren en gran parte integradas en una misma aplicación. Es decir, que para realizar una operación sobre el código fuente, no sea necesario tener que cambiar aplicación. Por ejemplo, si se necesita correr un analizador de código fuente éste debería estar integrado al mismo programa en el cual se está editando. Este hecho que puede parecer trivial no deja de ser importante, en la revisión de las herramientas hechas en el capítulo anterior la mayoría eran programas stand-alone o independientes. Una de las características de estas herramientas reside en la complejidad de las mismas. Realizar este tipo de proceso utilizando

"Buscar y Remplazar" (Search and Replace) no es suficiente. Un ejemplo de esto lo veremos en la Figura 4.5 en la cual se ha configurado a la herramienta de edición del presente documento (Latex) para mostrar código fuente Fortran haciendo que las palabras reservadas sean subrayadas. Dado que en Fortran es posible declarar una variable con el mismo nombre que las instrucciones del lenguaje (no existen el concepto de palabras reservada), la herramienta de edición (Latex) no ha podido reconocer cuáles palabras representaban la instrucción IF y cuáles la variable if.

```
        ip2=ip1
        twopi=twopic
        if =1
        if ( is .eq. 2 ) twopi=twopic
200     if ( ip2 .ge. ip4 )goto 480
        if = if +1
        if cur = ifact ( if )
        if ( ifcur .ne. 2 ) goto 120
        if ( ip2 .gt. ip4 ) goto 120
        if ( ifact (if +1) .ne. 2 ) goto 120
        if = if +1
120     ip3=ip2 + ifcur
        if ( if ) 120 ,200 ,480
        goto (62,42,43,62,404,45,62,62,62),I
```

Figura 3.11: Código Fortran Listado con el Paquete de Latex Lslisting en el Cual no Pueden Diferenciarse las Instrucciones del Lenguaje de los Identificadores de Variables

3.4. Las Cuatro Fases

En el proceso de desarrollo guiado por el cambio que se propone en este trabajo existen claramente cuatro fases bien definidas. Estas son:

- **La Comprensión**
- **La Transformación**
- **La Verificación**
- **La Retroalimentación**

A continuación se describirán las cuatro Fases.

3.4.1. La Fase de Comprensión

Esta fase está destinada a comprender el código fuente con el cual se está trabajando a partir de la aplicación de diversas herramientas destinadas a tales fines. Ejemplos del tipo de información que estas herramientas deberían proporcionar son: tabla de símbolos; árbol estático de llamadas; diversas métricas que proporcionen información de cuáles son las rutinas más complejas, las más largas, las más llamadas, etc; cobertura del código; entre otros. En el caso más específico de Fortran: áreas de COMMON BLOCKS, utilización de características obsoletas, cantidad de ENTRY POINTS dentro de las rutinas, cantidad y tipos de saltos de la instrucción GOTO (condicionales hacia adelante, condicionales hacia atrás, incondicionales). A partir de esta fase de comprensión del código fuente se obtendrá una lista de cambios a aplicar al mismo.

3.4.2. La Fase de Transformación

En esta fase se procederá a aplicar las transformaciones necesarias al software para llevar a cabo un proceso de mejora del mismo. Estos cambios se realizarán a través de herramientas automáticas de reestructuración de código fuente que deberán estar integradas y ser proveídas por el entorno de programación. Así como en el proceso de desarrollo guiado por los tests éstos son fundamentales, en un proceso guiado por el cambio/transformaciones las herramientas de desarrollo también lo son. En esta fase también se construirán los tests necesarios para la verificación del cambio.

3.4.3. La Fase de Verificación

En esta fase del proceso se verifican que los cambios hayan sido aplicados en forma correcta. Cabe destacar que la descripción de los procesos, actividades y tareas de esta fase quedan sujetos a toda una línea de investigación que excede a

este trabajo. En el caso del software científico la verificación conlleva además una validación numérica de los resultados del programa.

3.4.4. La Fase de la Retroalimentación

En esta última fase (que todo proceso debería tener) se realiza el trabajo de recabar información que pueda servir en la siguiente iteración del proceso. Esta fase se conoce como lecciones aprendidas. Básicamente se recolectan métricas e información relevante que pueda contribuir a mejorar el proceso en la próxima iteración. Y además, puede o no alimentar la lista de cambios a realizar incrementado dicha cantidad para las siguientes iteraciones.

3.5. El Flujo de Trabajo

A continuación se hará una descripción detallada de cuáles son los pasos propuestos para el flujo de trabajo.

3.5.1. Establecer una versión inicial del código fuente

En esta fase se fijará el punto de partida para el comienzo de la transformación y aplicación del cambio definido en el paso anterior. En el caso de encontrarse código fuente preexistente, dicha versión se fijará como punto de partida. Se estaría trabajando de hecho con código heredado. En el caso en que no exista implementación alguna, se aplicarán las técnicas de elicitación de requerimientos y diseño para obtener una versión inicial de los cambios que hay que aplicar teniendo en cuenta en este caso que el punto de partida sería la inexistencia del software y el cambio a aplicar sería el análisis y el diseño de los requerimientos y la funcionalidad.

3.5.2. Transformar el código fuente

Como primera tarea de este paso se definirán las pruebas para poder verificar que las transformaciones a efectuar hayan sido realizadas con éxito. A partir del

cambio definido en el paso anterior se llevará a cabo la transformación de código fuente en caso de que exista. De no existir dicho código, se transformará el diseño en la implementación del nuevo código fuente. Dichas transformaciones serán aplicadas utilizando herramientas automatizadas de reestructuración de código fuente integradas al entorno de desarrollo para el primer escenario, y para el segundo escenario (código inexistente) se utilizará el editor del entorno integrado de desarrollo para implementar el nuevo código fuente.

3.5.3. Verificar el código fuente obtenido

En esta fase se ejecutarán todos aquellos tipos de tests necesarios que han sido construidos en el paso anterior para verificar que la funcionalidad del código fuente transformado cumpla con los requerimientos iniciales. De fallar alguno de dichos tests en esta fase, se volverá al paso anterior.

3.5.4. Validar los resultados numéricos

Este paso, que puede ser opcional en el software que no requiera de validación numérica, está destinado a contrastar los resultados obtenidos con los esperados desde el punto de vista de los resultados numéricos. Si bien esto puede realizarse mediante testing también puede ser requerida una fase específica de validación numérica.

3.5.5. Aceptar/Rechazar el cambio en base a los resultados numéricos

Una vez que los últimos dos pasos han arrojado resultados favorables se contrastarán con los criterios de aceptación fijados en esta misma fase para fijar una nueva versión del producto.

3.5.6. Documentar

Este paso es utilizado para generar la documentación necesaria acerca de la transformación aplicada para hacerla disponible en todo momento. La documen-

tación también puede ser realizada mediante la utilización de herramientas automáticas y dentro del mismo código fuente del programa.

3.6. Resumen

En este capítulo hemos abordado la problemática del desarrollo de software desde la perspectiva del cambio y la transformación, dos aspectos esenciales del software [38] enfocándonos en un proceso guiado por las transformaciones que permita encarar desarrollos con código existente y heredado como así también con código que se construye desde cero. El ciclo propuesto para las iteraciones está íntimamente relacionado con el ciclo de Walter Shewhart, padre de los conceptos de Mejora Continua, Six-Sigma, proceso iterativo e incremental, entre otros [129]. Para ello se han propuesto una serie de fases que debieran llevarse a cabo a lo largo del mismo.

Capítulo 4

La Comprensión: Análisis de Código Fuente y Métricas para Fortran

En el capítulo se describirá una de las fases propuestas en el proceso de Desarrollo Dirigido por el Cambio, la fase de comprensión. Esta fase está enfocada en la obtención de conocimiento sobre el código fuente con el que se está trabajando, pues no es posible transformar algo que no se comprende. En el capítulo, además, se describirá la implementación de herramientas que pueden ser utilizadas en esta fase y que han sido integradas a un entorno de desarrollo.

4.1. La Comprensión del Código Fuente Heredado Fortran

El software heredado escrito en Fortran presenta ciertas características que lo hacen complejo de entender. Algunas de ellas son:

- Fortran es un lenguaje en el cual, en su formato fijo, los espacios en blanco son irrelevantes, de hecho el uso de VAR1 como variable es idéntico a VA R1, es decir los espacios en blanco no alteran al programa. En la historia

del lenguaje este hecho ha producido curiosos incidentes que han quedado como anécdotas en la historia de la Ciencias de la Computación, ver sección 3.3 del estándar de FORTRAN77 [84] "Blank characters preceding, within, or following a statement do not change the interpretation of the statement, except when they appear within the datum strings of character constants or the H or apostrophe edit descriptors in FORMAT statements. However, blank characters do count as characters in the limit of total characters allowed in any one statement". Por ejemplo, la sonda espacial que llegó a Venus fue Mariner II. Uno podría preguntarse qué sucedió con el Mariner I que no llego a destino. El código que guiaba el ascenso del cohete que llevaba a la sonda tenía escrito "`DO 5 k=1.3`". Debido a la característica antes mencionada de Fortran, esta instrucción fue interpretada correctamente por el compilador como: "`DO5 K= 1.3`", claramente la intención del programador era escribir "`DO 5 K = 1, 3`". El resultado de tal error fue que a los 294.5 segundos del despegue se tuviera que dar la orden de destrucción de la nave [165], este bug le costó a la NASA unos 18.5 millones de dólares aproximadamente, ver Figura 4.1 línea 5.

```
1          ...
2        CALL BESJ(X, 0, B0, EPS, IER)
3        IF (IER .EQ. 0) GOTO 10
4     20 CONTINUE
5        DO 5 K = 1. 3
6        T(K) = W0
7        Z = 1.0/(X**2)*B1**2+3.0977E-4*B0**2
8        D(K) = 3.076E-2*2.0*(1.0/X*B0*B1+3.0977E-4*
9      *(B0**2-X*B0*B1))/Z
10       E(K) = H**2*93.2943*W0/SIN(W0)*Z
11       H = D(K)-E(K)
12    5  CONTINUE
13       ...
```

Figura 4.1: Código Fortran Original de la sonda Espacial Mariner I

- En Fortran no existen las palabras reservadas [8] al igual que en PL/1, un ejemplo válido podría ser:

  IF ELSE THEN THEN=ELSE ELSE ELSE=THEN

  Esta porción de código fuente es perfectamente legal y funcional, incluso en

la actualidad.

- Mucho código fuente escrito tiene décadas.
- Los autores de dichos programas en gran mayoría no han utilizado métodos y técnicas modernas de programación. Incluso muchos de estos programas han sido creados cuando tales técnicas aún no existían. Pues Fortran antecede a la programación estructurada y a la programación orientada a objetos, aunque las ha ido asimilando en su histórica evolución.
- El lenguaje mantiene vigente ciertas estructuras sintácticas obsoletas que han sido eliminadas de la mayoría de los lenguajes modernos de programación, cuyo uso además están abolidos por las modernas técnicas de programación: IF aritmético, GO TO computado, múltiples puntos de entrada a rutinas, etc.
- Gran dispersión de herramientas no integradas a los entornos de programación.
- Los fabricantes de compiladores han admitido variaciones no especificadas en el estándar, como por ejemplo permitir que en código FORTRAN 77 los comentarios del estilo Fortran 90 sean válidos.

Debido a estos motivos particulares y otros más generales, estrechamente ligados con la esencia misma del software, es que la comprensión del código fuente heredado escrito en Fortran consume mucho tiempo y recursos. Es posible definir cierto código fuente escrito en Fortran como código "eyectable", es decir código que hace por sí mismo que el programador sienta que su mirada es eyectada fuera del programa.

En esta fase del proceso de desarrollo es necesario enfocarse en la extracción de información sobre el código fuente con el que se está trabajando. Para ello, es de fundamental importancia contar con un conjunto de herramientas de análisis de código fuente que proporcionen dicha información de manera precisa, clara y rápida. Por consiguiente, en esta etapa del proceso las herramientas de análisis

de código fuente son primordiales y de gran importancia para la comprensión del mismo.

4.2. Marco de Trabajo e Implementación

Uno de los ejes centrales de este trabajo de investigación está basado en la integración de varias herramientas bajo un mismo ambiente de desarrollo. Si bien el hecho de tener disponibles herramientas integradas puede ser considerado trivial, en el capítulo 2 se han descrito las herramientas que están disponibles para el desarrollo de software en Fortran, y la gran mayoría, por no decir su totalidad, no están integradas a ningún entorno de desarrollo moderno.

En 2005 bajo el patrocinio de la Universidad de Illinois, se desarrolló un plug-in para para Eclipse [88] que resultó en un entorno de desarrollo integrado (IDE) multiplataforma basado en Eclipse llamado Photran [1]. En la actualidad, este entorno de código abierto integra una serie de poderosas características, todas en una misma herramienta: edición de código fuente, compilación, depuración, navegación del código fuente, entre otras. Utiliza la herramienta *make* para compilar, con lo cual virtualmente puede utilizar cualquier compilador Fortran disponible en el mercado. También es posible integrar los errores de compilación de cada compilador a la herramienta, lo que permite obtener dichos errores en forma de texto con el agregado que éstos son marcados en el código fuente.

Photran fue pensado desde su creación como una herramienta de reestructuración en la cual dichas transformaciones son implementadas como refactorizaciones gracias a una infraestructura creada especialmente para ello. En su versión 6.0 (liberada en junio de 2010) ya permitía aplicar unas 16 refactorizaciones, en la versión 8.2 (liberada en junio de 2014) 39 transformaciones implementadas como refactorizaciones estaban disponibles para ser utilizadas.

Dentro de la arquitectura de la herramienta el componente más complejo es el paquete `org.eclipse.photran.core.vpg` que contiene básicamente toda la infraestructura para las transformaciones de código fuente: el AST (Abstrac Syntax Tree) y sus nodos, el Parser de Fortran, el preprocesador de Fortran (que maneja

los INCLUDE), el VPG o Virtual Program Graph, el motor de refactorización de código fuente y las refactorizaciones entre otros. Photran está compuesto por unas 250 mil líneas de código de las cuales 100 mil pertenecen a la infraestructura de refactorización y dentro de estas, 10 mil pertenecen a las transformaciones de código fuente.

Una herramienta que necesita realizar el tipo de análisis sobre el código fuente de un programa no puede absolutamente basarse en búsqueda y reemplazo (Search and Replace) o en cortar y pegar (Cut and Paste). Para ello el enfoque que se sigue es el mismo utilizado en los compiladores. Se basa en ASTs [127] reescribibles [171]. Un AST o Árbol de Sintaxis Abstracto es según [127] una estructura en la cual cada nodo intermedio del árbol representa a una construcción que aparece en el código fuente, los nodos hijos representan las componentes significativas de tal estructura, ver ejemplo de la Figura 4.2 de una oración simple en español y en la Figura 4.3 de una instrucción Fortran IF THEN ELSE.

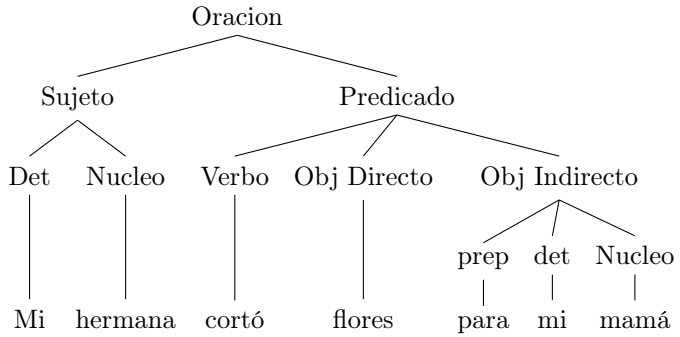


Figura 4.2: Ejemplo de AST de una Oración en Español

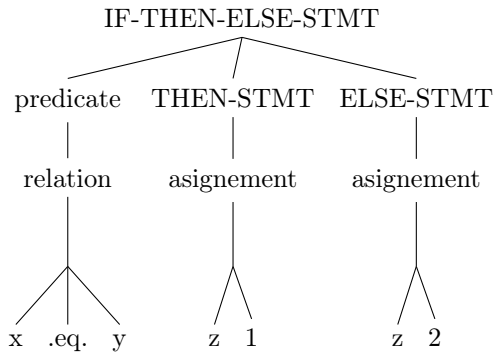


Figura 4.3: Ejemplo de AST de una Instrucción IF THEN ELSE

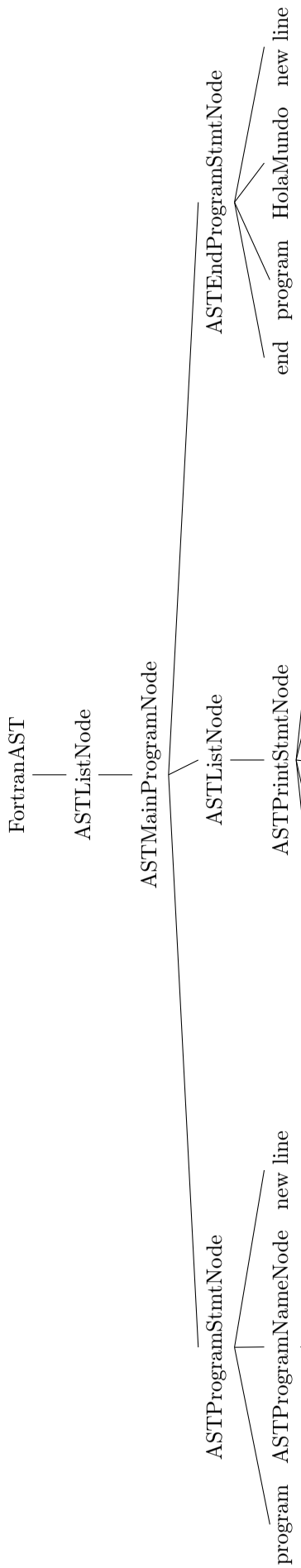


Figura 4.4: Ejemplo de un AST de la Vida Real Para un Programa Fortran de 1 Línea

Queda claro que en los ejemplos anteriores (triviales) el armado del AST puede ser relativamente sencillo. Sin embargo, dicha estructura en la vida real dista mucho de ser sencilla. Por ejemplo véase un pequeño programa de la Figura 4.4:

```
Program HolaMundo
    print *,"Hola Mundo"
End Program HolaMundo
```

Figura 4.5: Código Fortran Trivial

Para tener una idea, construir una infraestructura capaz de realizar este tipo de análisis se necesitan aproximadamente unas 90 mil líneas de código fuente y además las transformaciones propiamente dichas implican unas 10 mil líneas de código adicionales [170]. En el caso específico de Fortran cabe destacar que dadas las características del lenguaje no es sencillo generar dicha infraestructura (las distintas variaciones entre el formato fijo y el formato libre). Por ejemplo el Analizador Léxico o Lexer de Fortran posee tantas variantes que es más complejo generarlo con una herramienta tipo Flex, es decir automáticamente, que escribir uno manualmente.

4.2.1. La Infraestructura

El esquema general utilizado en este trabajo para la extracción de información de un programa escrito en Fortran es el siguiente:

1. A partir del código fuente del programa se utiliza un Lexer y posteriormente un Parser para generar un AST. Ambos son proporcionados por la herramienta.

2. Una vez generado el AST, se recorre el árbol normalmente implementando un patrón Visitor o Caminante o Visitante [92] para recorrer cada uno de los nodos del árbol de sintaxis en búsqueda de la información necesaria.

3. Una vez recorrido el árbol la información se transfiere a la Interfaz de Usuario (UI) de la herramienta para su visualización.

Estos son básicamente los pasos a seguir para recabar la información necesaria sobre el código fuente, en la Figura 4.6 se ve el código fuente, la Figura 4.7 representa al AST armado tras el análisis del programa y finalmente la Figura 4.8 se muestra un esquema de cómo se recorre el AST obtenido. En el proceso de construcción de la Figura 4.4 la complejidad del código Latex para armar dicha imagen requirió de la aplicación del método descripto anteriormente. Se debió armar una función Java que recorriera el AST y generara el código Latex necesario para mostrar la estructura de árbol que se ve en la imagen. En la Figura 4.9 se puede apreciar el código fuente Latex necesario para la construcción de la imagen, por ello se decidió aprovechar la infraestructura de Análisis de código fuente para armar la imagen a través de los pasos que anteriormente se detallan. Para ello se creó una clase Java que puede verse en la Figura C.1, del Apéndice C.

```
PROGRAM HolaMundo
PRINT *,"Hola Mundo"
END PROGRAM HolaMundo
```

Figura 4.6: Código Fuente a Ser Analizado

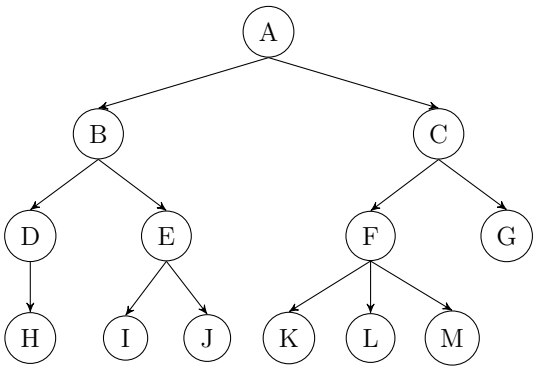


Figura 4.7: AST obtenido por el Lexer y el Parser de Photran

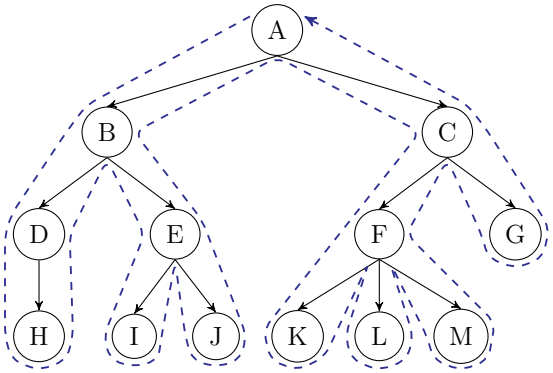


Figura 4.8: Recorrer la Estructura del AST Recolectando la Información

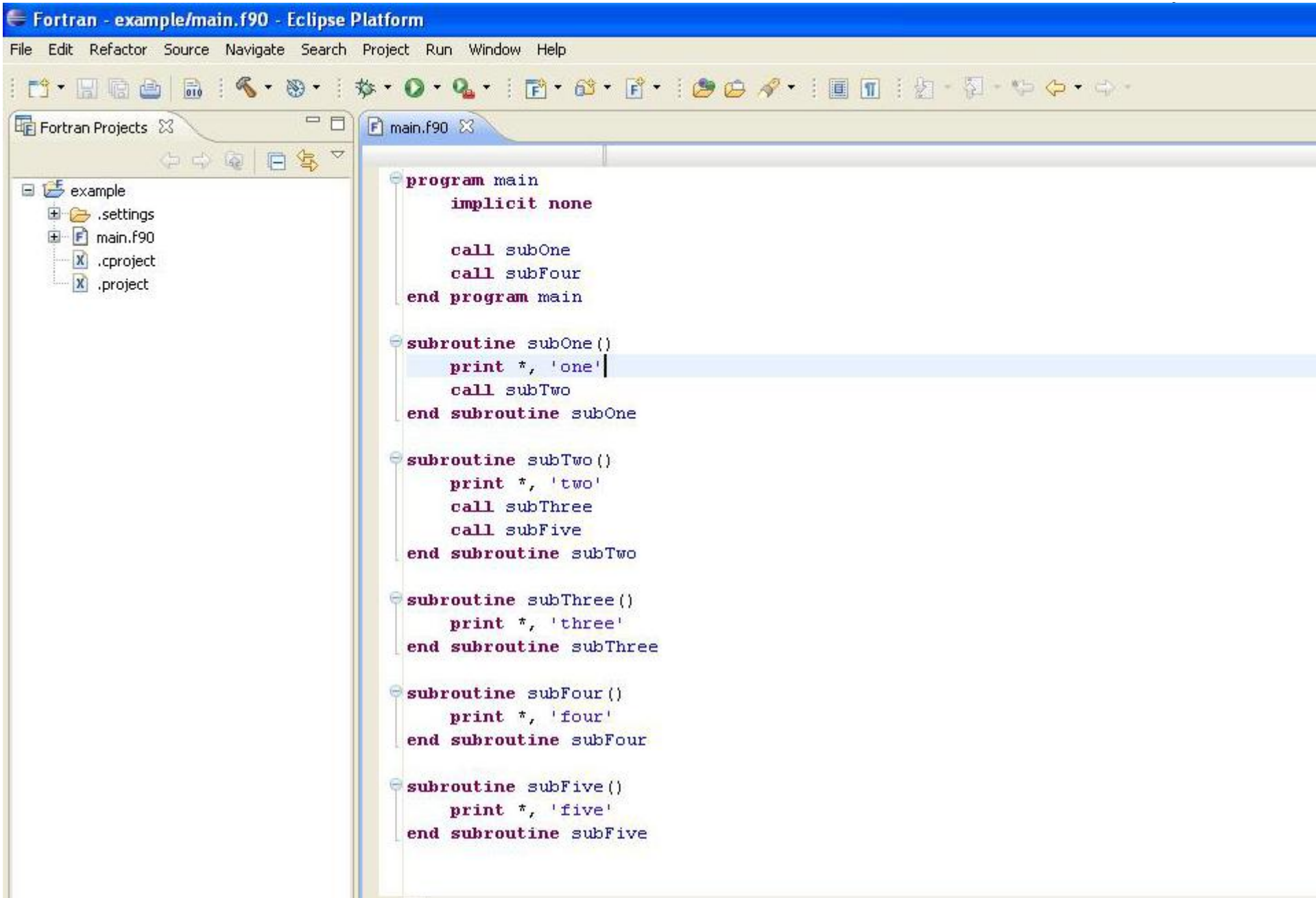
```
\Tree[ .FortranAST
         [ .ASTListNode
             [ .ASTMainProgramNode
                 [ .ASTProgramStmtNode
                           [ .program ]
                           [ .ASTProgramNameNode
                               [ .HolaMundo ] ][ .{new line} ]
                           ]
                 [ .ASTListNode
                     [ .ASTPrintStmtNode
                         [ .Print ]
                         [ .ASTFormatIdentifierNode
                            [ .* ]
                         ]
                         [ ., ]
                         [ .ASTOutputItemListNode
                            [ .ASTStringConstNode
                                [ .{"Hola Mundo!"} ]
                            ]
                         ]
                         [ .{new line} ] ] ]
                 [ .ASTEndProgramStmtNode
                     [ .end ][ .program ][ .HolaMundo ][ .{new line} ]
                     ]
               ]

          ]
  ]
```

Figura 4.9: Comando de Construcción del AST de la Figura 4.4

4.3. Análisis Estático

A partir de la implementación de los pasos anteriormente mencionados se han construido una serie de herramientas para realizar diversos tipos de análisis estático de código fuente Fortran, éstas han sido integradas a Photran. Entre una de las más importantes está la generación del árbol estático de llamadas, el cual muestra cómo se llaman las rutinas unas a otras. Esta herramienta se integró al entorno de desarrollo siendo construida como una "View", componente integrante de la interfaz de usuario de Eclipse, en la Figura 4.12 y la Figura 4.10.

Para la implementación de esta vista de Eclipse en primer lugar se crea un modelo, se recorre el AST del programa implementando un patrón visitante que recorra el árbol de sintaxis específicamente en los nodos que representan las declaraciones de funciones y procedimientos (ASTFunctionSubprogramNode y ASTSubroutineSubprogramNode) guardándolos en una lista. A su vez se visitan los nodos de llamadas a rutinas (ASTCallStmtNode), y de esta forma se recolectan llamadores y llamados. Para implementar la vista se utiliza esta información recolectada con la infraestructura que provee Photran simultáneamente con Zest, las herramientas de visualización de Eclipse, un conjunto de componentes para visualización construidos específicamente para el IDE. Un aspecto interesante de esta implementación es que, fuera del componente de Interfaz de Usuario, el árbol de llamadas puede ser fácilmente construido a partir del AST del código fuente del programa, ya que no pueden existir dudas de quiénes son los llamadores (callers) y los llamados (callees) por la naturaleza misma de la fuente de la información. Queda claro que sería poco preciso en un lenguaje con las características de Fortran (y posiblemente en cualquier otro) intentar hacer esto, por ejemplo, con expresiones regulares. Pues dada la definición de AST cada uno de los datos obtenidos ha sido recavados de una representación abstracta del código fuente.


Fortran - example/main.f90 - Eclipse Platform
File Edit Refactor Source Navigate Search Project Run Window Help
Fortran Projects
example
.settings
main.f90
.cproject
.project
main.f90
program main
implicit none
call subOne
call subFour
end program main
subroutine subOne()
print *, 'one'
call subTwo
end subroutine subOne
subroutine subTwo()
print *, 'two'
call subThree
call subFive
end subroutine subTwo
subroutine subThree()
print *, 'three'
end subroutine subThree
subroutine subFour()
print *, 'four'
end subroutine subFour
subroutine subFive()
print *, 'five'
end subroutine subFive

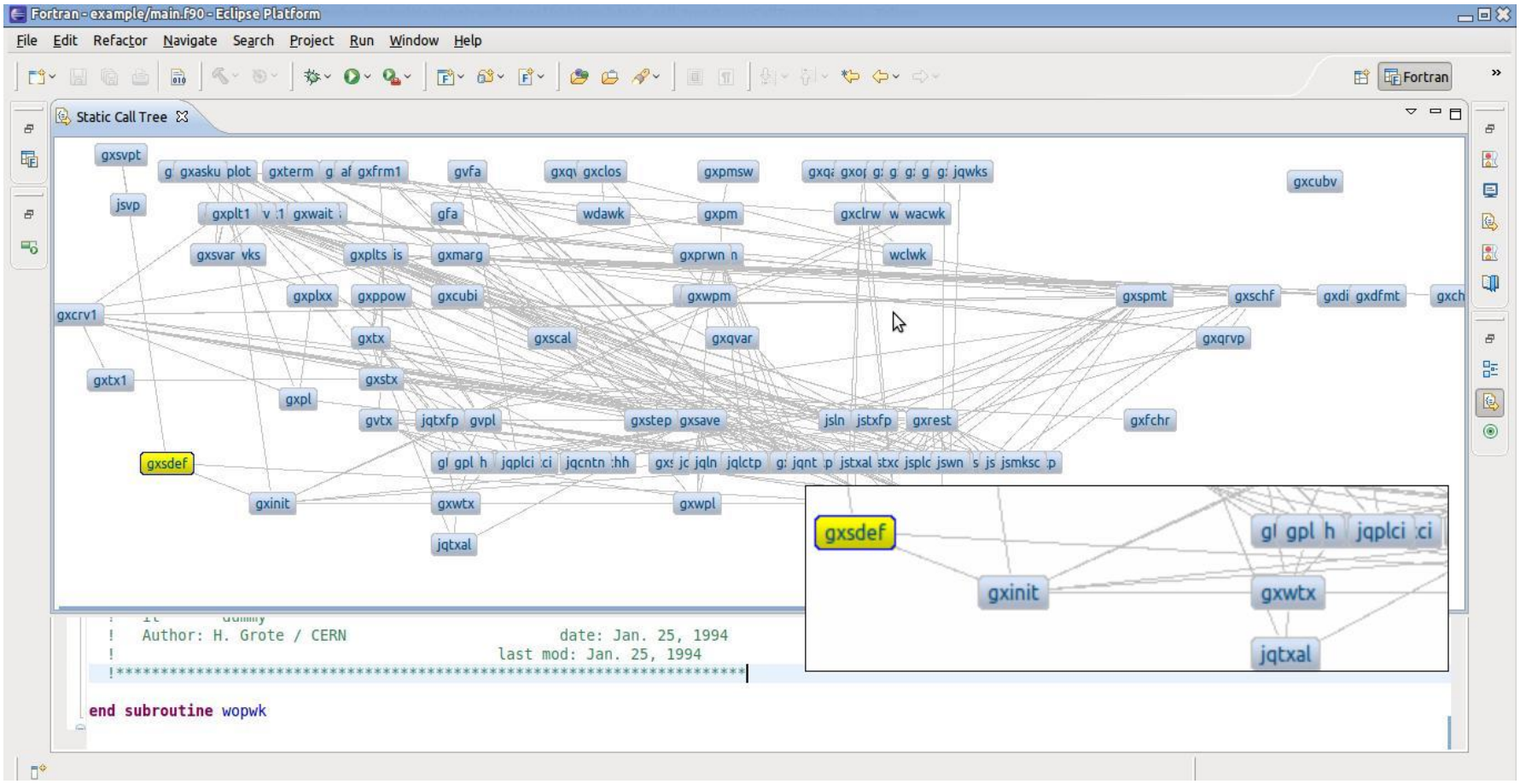


Figura 4.11: Implementación e Integración del Árbol de Estático de LLamadas

4.4. Métricas para Fortran

Haciendo referencia a las cuatro características esenciales del software [38] siempre se ha buscado definir medidas que permitan obtener información sobre este producto. Algunas de estas medidas específicas de Fortran se listan a continuación:

- Líneas de Código Fuente (LOC - Source Lines Of Code): LOC, Logical Executable LOC (LELOC), Non Blank Non Comments LOC (NbNcLOC), por archivo, por módulo y por subprograma.
- Metricas relacionadas con Archivos (File-related metrics): número de archivos, número de subrutinas y funciones por archivo.
- Complejidad Ciclomática de las rutinas [146] como se detalla en [22].
- Métricas relacionadas con Módulos: numero de instrucciones USE (identificando aquellas que tienen o no la cláusula "Only"), número de subprogramas públicos y privados (diferenciando entre funciones y subrutinas),número de datos de módulos (variables de módulos).
- Medidas relacionadas con los datos: cantidad de instrucciones COMMON, DATA, BLOCK DATA y EQUIVALENC, uso de parámetros (número de parámetros por rutina,numero de parámetros con Intent In/Out/InOut), número de datos locales y declaraciones de arreglos.
- Instruccion Include de Fortran y líneas del preprocesador de C: número de instrucciones Include de Fortran, número de lineas del preprocesador de C (directivas, macros, pragmas, errores, entre otros)y número de directivas de OpenMP.
- Medidas relacionadas con las iteraciones: número de instrucciones DO, nivel de anidamiento de las instrucciones DO (1-10), número de instrucciones DO en formato moderno (DO... END DO format), número de instrucciones DO en formato antiguo (etiquetado, sin END DO).

- Relacionadas con las instrucciones GO TO, instrucciones borradas o marcadas como obsoletas: número de Shared DO loops, número de instrucciones GO TO, número de instrucciones IF aritméticos , número de instrucciones assigned GO TO, número de instrucciones GO TO computadas, número de instrucciones ENTRY, número de instrucciones PAUSE, y número de instrucciones statement Function.

- Operaciones de Entrada y Salida: Número de instrucciones PRINT, Número de instrucciones READ, Número de instrucciones REWIND y Número de instrucciones WRITE.

- Otro tipo de característica de Fortran como número de etiquetas, número de IMPLICIT, número de EXTERNAL, número de instrucciones FORMAT, número de instrucciones, número de instrucciones IF, etc.

El mantenimiento de largos y complejos programas requieren de herramientas de ingeniería de software que ayuden a los programadores a realizar dichas tareas, sin embargo son muy escasas las herramientas avanzadas para el desarrollo de software para Fortran. La mayoría de estos programadores trabajan solamente con un editor de texto, compilador, debugger y quizás con alguna herramienta de análisis de performace. Mientras que los ingenieros de software que trabajan en lenguajes como Java o C# tienen acceso a entornos de desarrollo, herramientas de análisis estático, herramientas de refactorización, herramientas de testing, herramientas de comprensión de código, entre otras. Todas estas integradas en el mismo entorno de desarrollo. Este tipo de herramientas también podrían estar a disposición de los programadores Fortran [153].

Todas las métricas anteriormente listadas fueron exitosamente desarrolladas e integradas a Photran, como parte de este trabajo de investigación [152, 153]. A continuación se centrará la atención en algunas de las famosas métricas de complejidad que son utilizadas en los IDEs modernos y han sido implementadas también para Fortran en este trabajo.

4.4.1. Métricas de Complejidad

Desde los años 70 ciertas métricas han sido utilizadas para recavar conocimiento del grado de complejidad del código fuente. Una de las más conocidas es la que publicara Thomas McCabe en 1976 [146]. A su vez en la misma época Maurice Halstead propuso otra medida de la complejidad conocida actualmente como Halstead Computational Science Metrics [104, 103, 105]. A partir de estos estudios varias métricas de complejidad han sido creadas y utilizadas en distintos trabajos, a continuación se describe la implementación de una herramienta que proporciona los datos de estas medidas para código fuente escrito en Fortran y lo integra a un IDE moderno como Eclipse.

Complejidad Ciclomática

En 1976 se publica un trabajo que intenta medir la complejidad interna de un algoritmo, esta medida de complejidad se define en el mismo como "el número ciclomático V(G) de un grafo G con v vértices, e aristas y con una componente conectada es $V(G) = e - n + p$ " donde n es el número de nodos del grafo correspondientes a sentencias del programa y p corresponde al número de componentes conexos, nodos de salida. Este número proporciona la cantidad de caminos posibles a través de un programa, en otras palabras la Complejidad Ciclomática mide la cantidad de decisiones lógicas en un trozo de código fuente. Lo más atractivo de ésta es que el mismo McCabe descubrió que podía fijarse un umbral para caracterizar a los programas, funciones, procedimientos o módulos. En esa época McCabe sugirió el número 10 como un umbral para definir a un programa como complejo [146, 213]. Existe una propuesta publicada posteriormente en 1997 de la Universidad Carnegie Mellon [83], ver tabla

En este trabajo se ha implementado la Complejidad Ciclomática. Si bien existen varias implementaciones distintas de esta métrica se ha seleccionado aquella propuesta por Basili en 1983 en [22], con la salvedad de que se supone un único punto de entrada por rutina o programa:

"For programs with unique entry and exit nodes, this metric is equivalent to

Cyclomatic Complexity	Risk Evaluation
1-10	Módulo simple con muy poco riesgo
11-20	Módulo medianamente complejo con riesgo moderado
21-50	Módulo complejo con alto riesgo
51 o mayor	Programa de alto riesgo y altamente inestable

Cuadro 4.1: Descripción de la Complejidad de Código Fuente Según la C.C.

one plus the number of decisions and in this work, is equal to the one plus sum of the following constructs: logical if, if-then-else, block-if, block if-then-else's, do loops,while loops, AND's, OR's, XOR's, EQV's, NEQV's, twice the number of arithmetic if's, n-1 decision counts for a computed Go To with n statement labels, and n decision counts for a case if with n predicates".

Halstead Software Science Complexity

Maurice Halstead definió una métrica para medir la complejidad computacional del código fuente derivada precisamente de este mismo. En un intento por demostrar "que un algoritmo puede tener una estructura que obedezca ciertas leyes físicas", la métrica propuesta es capaz de obtener información de un conjunto independiente de propiedades sobre algoritmos que no dependen del lenguaje en el que éste esté escrito [146, 213]. Para obtener su conjunto de métricas, ya que lo definido por Halstead no es una única métrica sino un conjunto de ellas, se basa en la distinción de dos tipos de entidades dentro de un programa: operadores y operandos. A partir de esta distinción se derivan una serie de propiedades:

- Número de los diferentes operadores que aparecen en el código fuente η_1 .
- Número de los diferentes operandos que aparecen en el código fuente η_2.
- El total de ocurrencias de operadores en el programa N_1.
- El total de ocurrencias de operandos que aparecen en el programa N_2.

A partir de este conjunto de propiedades que se obtienen de contar la cantidad de operandos y la cantidad de operadores que ocurren en el programa, se desprenden otras propiedades en término de aquellas definidas en [104, 103, 105]:

- Entidades Únicas (Unique Entities) o también conocida como Vocabulario del Programa (η): se define como el número de operadores únicos más el número de operandos únicos en el programa. Se calcula como $\eta = \eta_1 + \eta_2$

- Entidades Acumuladas, también conocida como Longitud del Programa (Program Length) N: es el número total de operadores más el numero total de operandos en todo el programa. Se obtiene de la siguiente forma $N = N_1 + N_2$

- Volumen del Programa (Program Volume) V: "consiste en el producto del total de ocurrencias de entidades en el programa, N, y el número de dígitos binarios que se requerirían para especificar cada entidad a lo largo de todas las distintas entidades" [104]. El volumen del programa se calcula como: N $\log_2 \eta$.

- Nivel (L): "tiene un valor máximo de 1, que se alcanzará cuando el número de distintos operadores sea el mínimo valor de 2, y ningún operando sea mencionado más de una vez "[104]. El nivel del programa se obtiene como $2/\eta_1 * \eta_1/N_2$

- Dificultad (D): se define como $1/L$.

- Esfuerzo (E): se define como $V * D$

Este conjunto de métricas con sus propiedades también se han implementado e integrado a Photran. Al igual que con la Complejidad Ciclomática de McCabe, existen variaciones en cómo obtener dichas métricas [41, 22], el enfoque seleccionado es el utilizado por [22].

4.4.2. El Índice de Mantenibilidad de Coleman

A medida que el mantenimiento de software ha ido desarrollándose, también sus prácticas lo han hecho. Muchos autores sostienen que el mantenimiento de software es el proceso con los más altos costos dentro del proceso de desarrollo.

En [37] se afirma "El costo total de mantenimiento de un programa altamente utilizado es del 40 % del costo total de desarrollo del mismo".

En 1979, los costos de mantenimiento rondaban el 67 % [221], en los 90 este número se aproximaba al 75 % [72], y finalmente a los comienzos de la década del 2000 ese valor ya se calculaba cerca del 90 % del costo total del proyecto [77]. Estos números demuestran dos aspectos importantes en lo que respecta al desarrollo de software, transformar software aplicando cambios es lo más costoso en el proceso de desarrollo y lo más común dentro del mismo. Hacia mitad de los años 90 Coleman et al. propusieron una métrica basada en tres dimensiones para medir el grado de mantenibilidad del software [54]:

- Estructura del Control
- Estructura de la Información
- Tipografía, nomenclatura y comentarios

Esta métrica propuesta es el resultado del análisis de diferentes modelos de regresión. "El modelo de regresión que pareció más aplicable fue un modelo polinómico de cuatro métricas, basado en las métricas de Halstead, la Complejidad Ciclomática de MaCabe, la cantidad de líneas de código y la cantidad de comentarios" [54]. Como resultado del estudio de Coleman el índice de mantenibilidad se define como:

$$171 - 3{,}2 * ln(HV) - 0{,}23 * (CC) - 16{,}2 * ln(LinesofCode)) + Comments$$

Donde HV = Halstead Volume y CC = Cyclomatic Complexity.

La implementación llevada a cabo en este trabajo es igual a la de Coleman pero ligeramente modificada:

$$171 - 3{,}2 * ln(HV) - 0{,}23 * (CC) - 16{,}2 * ln(LinesOfCode)) * 100/171$$

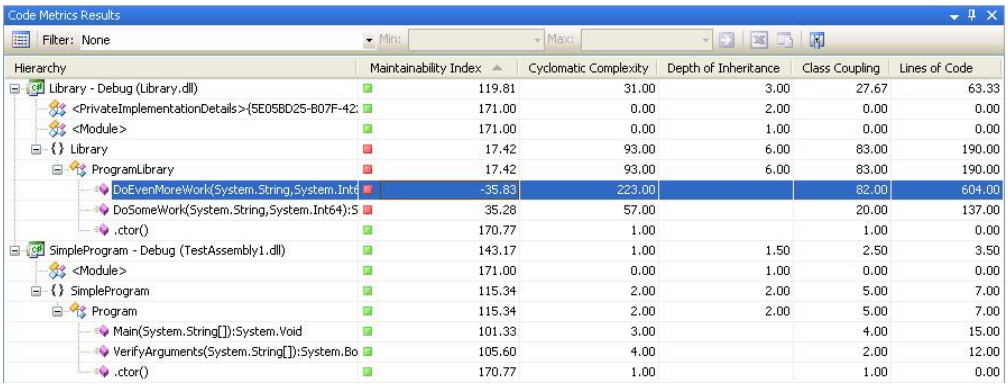
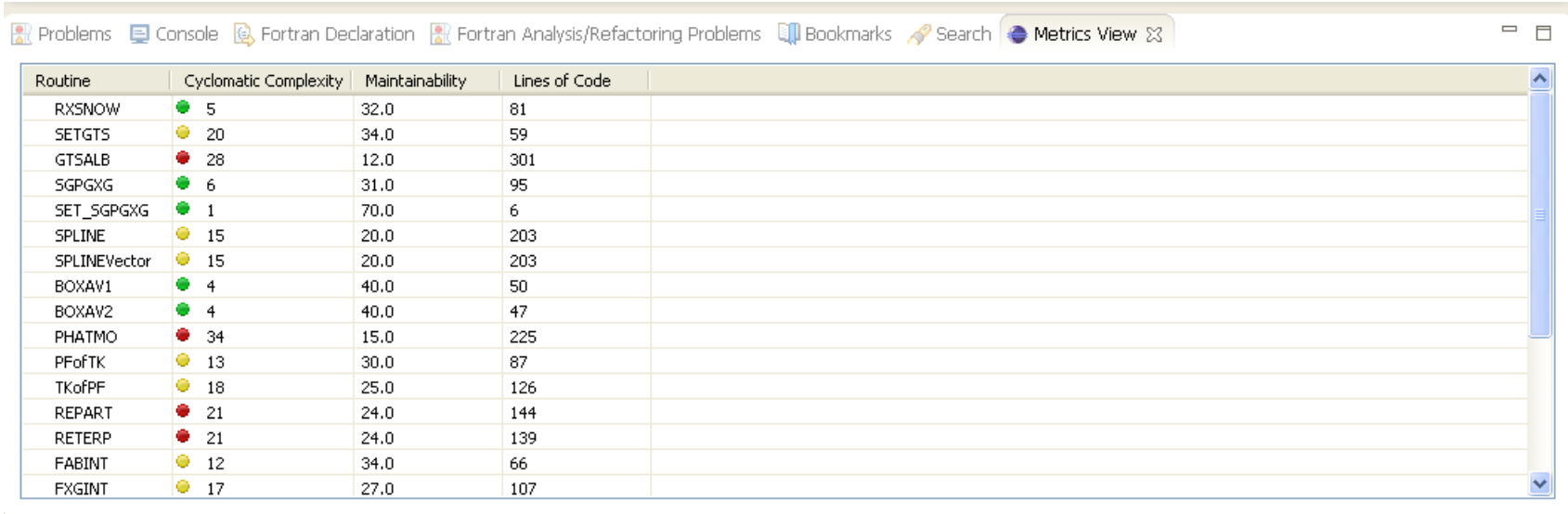


Figura 4.12: Métricas Integradas al IDE Microsoft Visual Studio Por Microsoft

Donde HV = Halstead Volume y CC = Cyclomatic Complexity. Cabe destacar que la utilizada es la misma métrica implementada en el producto Microsoft Visual Studio 2008 y versiones sucesivas:

Implementación

Todas estas métricas han sido implementadas teniendo en cuenta el proceso descrito en el 4.2.1. Una clase que implementa el patrón de diseño visitante (visitor) recorre el AST del programa que se encuentra en el editor activo en ese momento y a través del conteo definido para cada métrica se recolectan dichos datos para ser presentados a la UI del IDE. Dicha implementación fue agregada al entorno de desarrollo integrado Photran mediante la construcción de un componente de la interfaz de usuario de eclipse llamado Vista (View), que permite ver en el mismo momento en que se está editando el código fuente cuáles son los valores de estas métricas de complejidad. En las Figuras 4.13 y 4.14 puede apreciarse dicha integración. De esta forma se ha dotado de nueva funcionalidad a Photran comparada a la de los IDEs de lenguajes modernos como C#, C++, entre otros.



Routine	Cyclomatic Complexity	Maintainability	Lines of Code
RXSNOW	5	32.0	81
SETGTS	20	34.0	59
GTSALB	28	12.0	301
SGPGXG	6	31.0	95
SET_SGPGXG	1	70.0	6
SPLINE	15	20.0	203
SPLINEVector	15	20.0	203
BOXAV1	4	40.0	50
BOXAV2	4	40.0	47
PHATMO	34	15.0	225
PFofTK	13	30.0	87
TKofPF	18	25.0	126
REPART	21	24.0	144
RETERP	21	24.0	139
FABINT	12	34.0	66
FXGINT	17	27.0	107

Figura 4.13: Vista de Eclipse en la Cual se Muestran las Métricas Obtenidas.

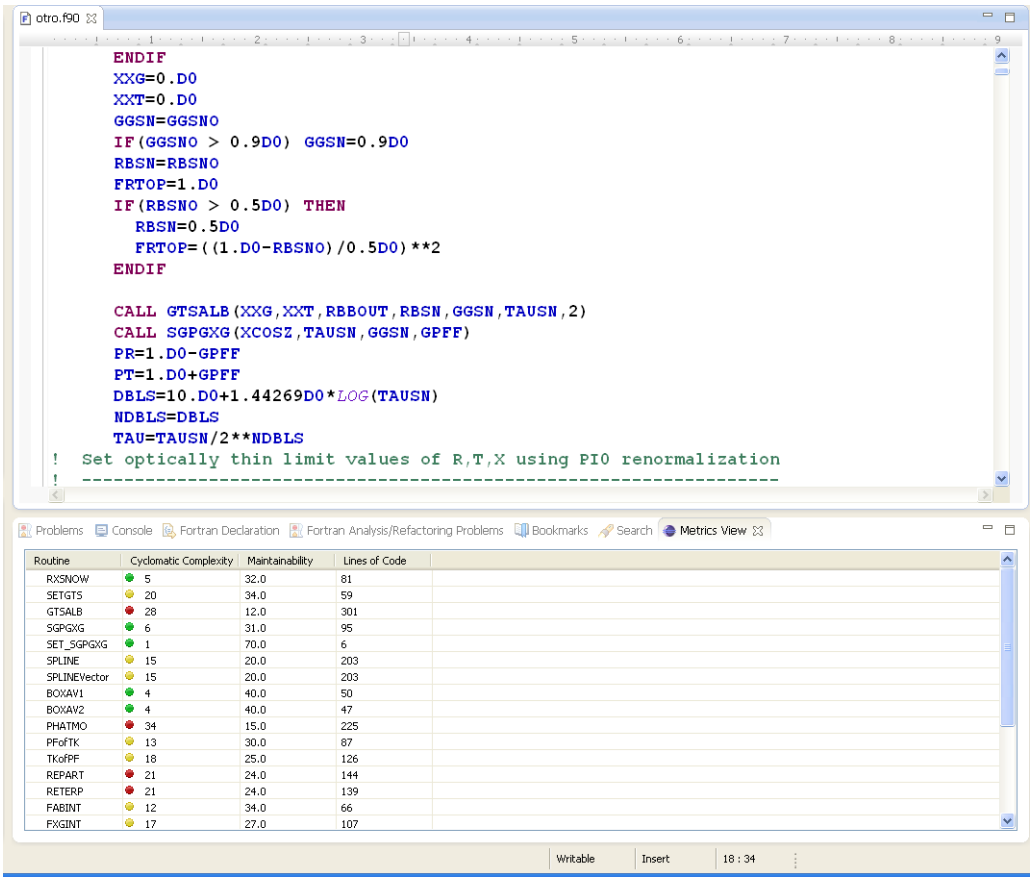


Figura 4.14: Integración de la Vista de Métricas en el Editor de Texto del Código Fuente.

4.5. Características Obsoletas del Lenguaje

Otra característica importante a la hora de tener en cuenta el software heredado escrito en Fortran es la utilización de construcciones del lenguaje que han sido declaradas como obsoletas, por ejemplo, la instrucción IF aritmética, la instrucción GO TO computado, por nombrar algunas (ver Apéndice B del estándar de Fortran 90 en adelante [9]). Alguna de estas construcciones son instrucciones de salto condicionales que rompen la estructura de un programa, dichas construcciones no deberían ser utilizadas en la actualidad. Además, si el objetivo es paralelizar alguno de estos programas la utilización de construcciones de salto dentro del código a paralelizar hacen que este proceso sea muy difícil de llevar a cabo. Siempre dentro del contexto de la comprensión del código fuente con el cual

se está trabajando se desarrolló una herramienta que lista la línea en la cual se encuentran algunas de las llamadas construcciones obsoletas del lenguaje Fortran. La misma también integrada al IDE, de forma tal de ser utilizada mientras se trabaja con el código fuente del programa. Para ello siempre siguiendo los puntos presentados en la sección 4.2.1 se procedió a la implementación de una clase java que implementará el patrón visitante o visitor [92] cuyo objetivo es el de detectar aquellos nodos en el AST del programa que definen a una de estas estructuras obsoletas, estos nodos son:

1. ASTGotoStmtNode
2. ASTAssignedGotoStmtNode
3. ASTArithmeticIfStmtNode
4. ASTDataStmtNode
5. ASTStmtFunctionStmtNode
6. ASTComputedGotoStmtNode
7. ASTProperLoopConstructNode

Por cada nodo se registra la línea y columna en que estas construcciones de lenguaje son utilizadas. Una vez que todo el árbol ha sido recorrido se muestra un informe en formato HTML en la interfaz de usuario de entorno de programación. El análisis se realiza por cada archivo de código fuente de un proyecto, ver la Figura 4.15. La integración al entorno de programación puede verse en la Figura 4.16.

Deleted and Obsolescent Features Analysis : balanc.f

The obsolescent features are those features of FORTRAN 77 that are redundant and for which better methods areavailable in FORTRAN 77 (See Fortran Standard Annex B).The detected obsolescent features are:

Arithmetic If Stmts found = 0
Assigned Go To Stmts found = 0
Computed Go To Stmts found = 0
Data Stmts found = 5

```
In GRIDCON at line : 766

In CREATE_CTLFILE at line : 2726

In MODEL_TIME_DEF at line : 3728

In MODEL_TIME_PRINT at line : 3857

In VGRID at line : 5156
```

Pause Stmts found = 0
Statement Functions found = 0
Shared Do Loop Stmts found = 0
Entry Stmts found = 0

Figura 4.15: Listado Emitido por la Herramienta para Mostrar las Características Obsoletas Utilizadas en el Programa

Figura 4.16: Integración del Listado de Características Obsoletas de Fortran

4.6. Análisis de COMMON BLOCKS

Otra característica interesante para realizar un análisis detallado del código fuente Fortran son los COMMON BLOCK o Bloques Comunes. Dado que en las primeras versiones de Fortran no existía el concepto de variable global, todas las variables son locales al ámbito en el cual son declaradas, los COMMON BLOCK fueron introducidos en el lenguaje en la versión de FORTRAN II [13]. Uno de los problemas de los COMMON BLOCK es que su definición debe copiarse en cada una de las rutinas que utilice alguna de las variables definidas en dicho COMMON BLOCK (COMMON /nombre/ lista-de-variables). Una de las características de estos bloques de datos es que sus variables no tienen por qué llamarse de la misma manera, ver ejemplo de la Figura 4.17. Ésta es una de las características entre otras que hacen que el análisis de COMMON Blocks sea muy complejo de realizar

a mano o a simple vista. Este hecho es de suma importancia en el momento de intentar realizar por ejemplo paralelización utilizando memoria compartida del código fuente, ya que el uso de variables globales a nivel de escritura no controlada por parte de alguno de los threads puede causar efectos no deseados.

```
----|----1----|----2----|----3----|----4----|----5----|
      PROGRAM MAIN
      INTEGER A
      REAL    F,R,X,Y
      COMMON /MICOMMON/ R,A,F
      A = -14
      R = 99.9
      F = 0.2
      CALL SUB(X,Y)
      . . .
      END

      SUBROUTINE SUB(P,Q)
      INTEGER I
      REAL    A,B,P,Q
      COMMON /MICOMMON/ A,I,B
      . . .
      END
```

Figura 4.17: Ejemplo de Utilización de un COMMON Block Renombrando sus Variables

Para el análisis de los COMMON Blocks se ha implementado e integrado una herramienta que permite estudiar todos los COMMON que se encuentran en un archivo de código fuente Fortran, en el cual se muestra para cada variable de un COMMON Block utilizado en ese ámbito lo siguiente:

- Nombre de la Variable.
- Especificación de Tipo.
- Especificación de Atributo.
- Tipo de acceso en ese ámbito: Lectura / Escritura / Lectura-escritura

Se ha incluido un ejemplo a partir del código fuente listado en la Figura 4.18 y la salida generada a partir del análisis se puede apreciar en la Figura 4.19. Cabe destacar que se ha implementado esta herramienta siempre siguiendo los

puntos presentados en la sección 4.2.1. Procediendo a la implementación de una clase java que implementara el patrón visitante o visitor [92] cuyo objetivo es el de detectar aquellos nodos en el AST del programa que definen los COMMON BLOCK y a partir de allí realizar el análisis correspondiente. Esta herramienta ha sido construida e integrada al Ambiente de Desarrollo Eclipse.

4.7. Resumen

En este capítulo se ha presentado la importancia de las herramientas de análisis de código fuente dentro de la etapa de comprensión de un poseso de desarrollo que está guiado por el cambio, pues no es imposible cambiar algo que no se comprende. Desde este punto de vista las herramientas de análisis de código fuente integradas en un entorno de desarrollo permiten agilizar este proceso. Por otro lado la infraestructura para la construcción de este tipo de herramientas se limitan a tres componentes: un Lexer + un Parser y un patrón visitante para recorrer el AST generado por el Parser.

```
----|----1----|----2----|----3----|----4----|----5----|----6----|----7--
      program tcommon
C _X(Here we use implicit declarations ... all REAL variables)
      common /abc/a,b,c
C
      call suba()
      call subb()
      call subc()
C
      write(6,'(1x,"a,b,c=",3f7.2)') a,b,c
      stop
      end
c=======================================================
      subroutine suba()
      common /abc/a,b,c
C _X(must declare 'abc' exactly the same in each routine)
      a = 1.0
      return
      end
c=======================================================
      subroutine subb()
      common /abc/a,b,c
C _X(all variables are visible to each routine that accesses 'abc')
      b = 2.0
      c = 3.0
      return
      end
c=======================================================
      subroutine subc()
C _X(Common source of problems, suppose implicitly)
C _X(declare 'C' variables complex ... yields)
C _X(different size common block)
      common /abc/a,b,c
      a=b+c
      return
      end
```

Figura 4.18: Código Fuente Fortran con Declaraciones de COMMON BLOCKS

Common Block usage for : common.f77
Display Common Block Analysis:common.f77

Scope: tcommon - Implicit Enabled - Default Implicit Spec: real (a-h), integer (i-n), real (o-z)			
Variable Name	Type Specification	Attribute Specification	Access Type in Scope
common block: /abc/			
a	Real	Implicit	R
b	Real	Implicit	R
c	Real	Implicit	R
Scope: suba - Implicit Enabled - Default Implicit Spec: real (a-h), integer (i-n), real (o-z)			
Variable Name	Type Specification	Attribute Specification	Access Type in Scope
common block: /abc/			
a	Real	Implicit	W
b	Real	Implicit	not used
c	Real	Implicit	not used
Scope: subb - Implicit Enabled - Default Implicit Spec: real (a-h), integer (i-n), real (o-z)			
Variable Name	Type Specification	Attribute Specification	Access Type in Scope
common block: /abc/			
a	Real	Implicit	not used
b	Real	Implicit	W
c	Real	Implicit	W
Scope: subc - Implicit Enabled - Default Implicit Spec: real (a-h), integer (i-n), real (o-z)			
Variable Name	Type Specification	Attribute Specification	Access Type in Scope
common block: /abc/			
a	Real	Implicit	W
b	Real	Implicit	R
c	Real	Implicit	R

Figura 4.19: Ejemplo del Listado HTML del Uso de las Variables de los COMMON BLOCKS

Capítulo 5

La Transformación: Transformaciones de Código

Tal vez el paso más importante en el Desarrollo Guiado por el Cambio sea exactamente la forma en que se aplican dichos cambios al código fuente. Al implementar un cambio dentro del software en funcionamiento se tiene que garantizar que la funcionalidad actual no será cambiada, a menos que el cambio sea justamente un cambio de funcionalidad. Este trabajo nos basará en cambios relacionados con la estructura interna del software y no con su funcionalidad. Haciendo especifico hincapié aquellos cambios que al ser aplicados ayudan en el proceso de paralelización de dichas aplicaciones. Por ejemplo la eliminación de características obsoletas del lenguaje, la eliminación de las instrucciones GO TO, entre otras. Un paso más ambicioso es el de construir transformaciones de código fuente capaces de paralelizar porciones de dicho código, por ejemplo mediante la utilización de bibliotecas como OpenMP.

5.1. Transformación de Código Fuente

Una transformación de código fuente automatizada puede dividirse claramente en tres etapas a partir de la obtención del AST de la porción de código fuente a

ser transformado o cambiado:

1. Precondiciones de la Transformación: En esta etapa se chequean las precondiciones o los requisitos necesarios, sobre el código fuente, para asegurarse que la transformación sea aplicable.

2. Aplicación de la Transformación: Esta etapa a su vez se divide en:

   a) búsqueda del punto del cambio dentro del AST.

   b) modificación del AST .

3. Postcondiciones de la Transformación: Condiciones que debe cumplir el programa una vez realizada la transformación.

Estas son las tres etapas necesarias en la aplicación de una transformación del código fuente.

De la misma forma que en la Sección anterior, la idea principal se sustenta en aplicar una transformación de código fuente mediante la modificación del AST del programa. Por ello todas las etapas de la transformación del código fuente, ya sea la verificación de las pre y post condiciones, o la etapa de transformación se sustentan en la utilización del AST y del patrón visitante ver Figuras 5.1, 5.2 y 5.3.

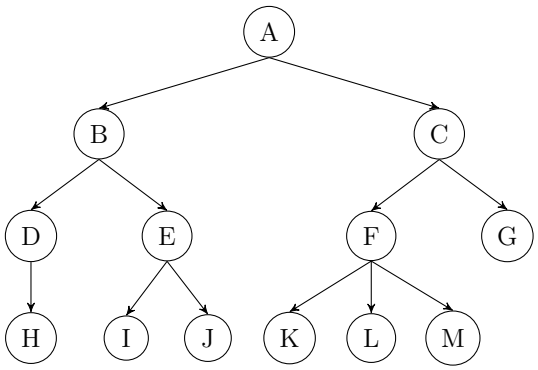


Figura 5.1: AST Obtenido por el Lexer y el Parser de Photran del Código Fuente

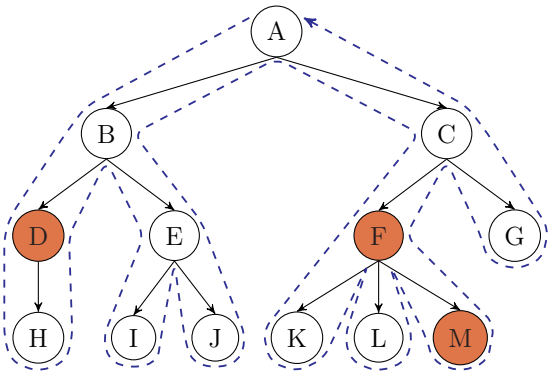


Figura 5.2: Recorrer la Estructura del AST Verificando las Pre-condiciones Mediante un Visitante

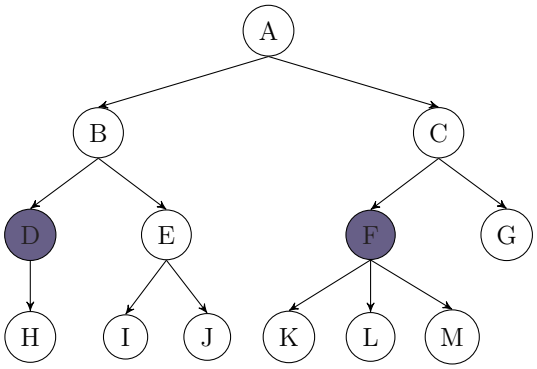


Figura 5.3: Transformar los Nodos del AST que Cumplan Dichas Pre-condiciones

En un trabajo anterior se definió un catálogo de transformaciones de código fuente para Fortran [149, 150, 174] el mismo ha ido evolucionando a través del tiempo y hoy posee esta lista de transformaciones:

1. **Transformaciones Para Mejorar La Presentación / Legibilidad**

   - Rename*: cambia el nombre de una variable, subprograma, etc.
   - Extract Local Variable: elimina una subexpresión dentro de una expresión mayor y se la asigna a una variable local.

*Implementada

- Extract Local Procedure*: extrae una secuencia de instrucciones de un procedimiento, las sitúa en una nueva subrutina y reemplaza las intrucciones extraídas con la llamada al procedimiento.
- Canonicalize Keyword Capitalization*: hace que todos los archivos de código fuente seleccionados tengan el mismo formato de Mayúsculas o Minúsculas.
- Standarize Statements*: Reescribe todas las declaraciones de las variables, por lo que sólo deja una declaración de variable por línea, y cada declaración contiene dos puntos dobles (::). Con ello se pretende hacer el código más legible.
- Convert Specification Statement to Declaration Attribute*: Sustituye las distintas declaraciones de especificación (DIMENSIÓN, PARÁMETROS, SAVE, etc.) si las hubiera, con los mismos especificadores pero contraídos en una única línea.

2. **Transformaciones Para Facilitar El Diseño/Cambio De Interfaces**

- Encapsulate Variable*: crea los métodos getters y setters para la variable seleccionada.
- Make Private Entity Public*: cambia la visibilidad de una variable de módulo o subprograma de privada a pública.
- Change Subprogram Signature: permite al usuario agregar, cambiar, extraer, reordenar, renombrar o cambiar de tipos de parámetros de una función o subprograma, actualizando la llamada de acuerdo a esto.
- Add Only Clause To Use Statements: crea una lista de símbolos que están siendo utilizados por un módulo y la agrega en la instrucción USE correspondiente.
- Move Entity Between Modules: mueve una variable de módulo o un procedimiento de un módulo a otro haciendo los ajustes correspondientes en la instrucción USE.

- Add Use of Named Entities To Module*: permite seleccionar una entidad en un módulo y agregarle la cláusula USE ONLY en el módulo seleccionado.
- Safe-Delete Internal Subprograms*: Este refactoring elimina aquellas subrutinas o funciones no utilizadas.
- Change Subroutine to Function: permite transformar una subrutina en una función.
- Change Subroutine Signature*: permite reordenar los parametros de una subrutina.
- Add Subroutine Parameter: este refactoring permite agregar parámetros a una subrutina y ajustar todas las llamadas correspondientes.

3. **Transformaciones que Evitan Prácticas Pobres de Programación en Fortran**

   - Remove Unreferenced Labels*: borra las etiquetas que no son utilizadas.
   - Remove Real Type Iteration Index*: elimina parámetros o variables de control de los ciclos DO no enteras.
   - Remove Reserved Words As Variables: renombra las variables con el mismo nombre que palabras reservadas de Fortran.
   - Introduce Implicit None*: agrega la instrucción Implicit None en un archivo y agrega las declaraciones explícitas para todas las variables que fueron declaradas previamente como implícitas.
   - Introduce Intent In/ Out*: agrega en las declaraciones de variables dentro de funciones o procedimientos la palabras reservadas Intent IN / Intent OUT según corresponda.
   - Remove Unused Local Variables*: elimina variables locales declaradas y nunca utilizadas.

- Minimize Only List: elimina símbolos que no están siendo utilizados de la lista ONLY en la instrucción USE.
- Make Common Variable Names Consistent: hace que todas las definiciones de una variable dentro de un bloque COMMON sean la misma.
- Delete Unused Common Block Variable: elimina las variables declaradas en los bloques COMMON nunca utilizadas.
- Add Dimension Statement: agrega la instrucción DIMENSION en las declaraciones de un array.
- Remove Format Statement Labels*: reemplaza los códigos de las instrucciones FORMAT en las declaraciones de las instrucciones READ/WRITE, en lugar de dejar la especificación del formato separada de la instrucción.
- Add identifier to END statement (eje. END SUBROUTINE FOO)*: Agrega el nombre del identificador según corresponda.
- Toggle End Name*: Esta transformación está diseñada para agregar o quitar automáticamente el nombre al final de una construcción sintáctica que puede ser nombrada, por ejemplo END SUB.

4. **Transformaciones que Eliminan Construcciones Antiguas, Obsoletas o No-Standard**

   - Replace Obsolete Operators*: reemplaza todas las ocurrencias de las operaciones de comparación del viejo estilo (como .LT. y .EQ.) con sus nuevos equivalentes (símbolos como $<$ y $=$).
   - Change Fixed Form To Free Form*: cambia el formato fijo de archivos Fortran por el Formato de estilo Libre.
   - Transform Character* to Character(Len =) declaration*: remplaza CHARACTER* con su equivalente CHARACTER(LEN =) en las declaraciones de strings.

- Remove Computed GO TO statement*: remplaza los GO TO computados con una construcción SELECT-CASE equivalente que puede contener o no los GO TO.
- Remove Arithmetic If statement*: remplaza las viejas instrucciones de tipo IF aritmético con su correspondiente instrucción IF.
- Remove Assigned GO TOs: elimina los GO TO asigandos.
- Replace Old Styles DO loops*: reemplaza las instrucciones DO Loop Continue (viejo estilo) con su equivalente DO Loop que finaliza con End DO.
- Replace Shared Do Loop Termination*: reemplaza los Shared DO loop Termination con la construcción equivalente Do Loop que finaliza con End Do.
- Transform To While Sentence: elimina los ciclos While creados a partir de instrucciones IF y GO TO.
- Move Common Block to Module: saca todas las declaraciones de un COMMON BLOCK particular, moviendo las declaraciones a un módulo e introduce la cláusula USE donde sea necesario.
- Move Saved Variables To Common Block: crea un COMMON BLOCK para todas las variables del tipo Saved de un subprograma.
- Data To Parameter*: cambia la declaración de un identificador de Data a Parámetro.

Transformaciones para Performance

Esta categoría actualmente tiene dos ejemplos de cómo las refactorizaciones pueden ser utilizadas para mejorar la performance preservando no sólo el comportamiento del programa sino también la legibilidad y la mantenibilidad del código. Éste es uno de los factores que suelen apartar la refactorización de la optimización.

1. Transformaciones para Performance Específicas

- Change To Vector Form*: reescribe un ciclo Do en su equivalente utilizando la notación vectorial de Fortran, permitiéndole al compilador hacer mejores optimizaciones.
- Make DO Loop Concurrent/Non-Concurrent (Fortran 2008): transforma un ciclo DO Loop en un ciclo Concurrente.

2. Transformaciones Para Ciclos

- Interchange Loops*: intercambia en un ciclo Loop el ciclo interno por el externo, en el caso que este intercambio pueda ser realizado esto permite optimizar el patrón de acceso a memoria y permite tomar ventaja de las técnicas de prefetching.
- Fuse Loops*: toma dos ciclos DO, normaliza sus subíndices y finalmente construye un único ciclo.
- Reverse Loop*: toma un ciclo cuyos subíndices se incrementan o decrementan, intercambia los valores iniciales y finales, y modifica la operación aplicada a la variable de control.

Todas aquellas transformaciones marcadas con (*) han sido implementadas e integradas al entorno de Programación Photran, alguna de estas implementaciones han sido realizadas como resultado de este trabajo. En la versión 8.0 que fue liberada en junio de 2014 la herramienta tenía disponible para su utilización 39 transformaciones de código fuente implementadas como refactorings.

5.2. Implementación de las Transformaciones de Código Fuente Fortran

Photran proporciona toda la infraestructura necesaria para la implementación de transformaciones de código fuente, ver Figura 5.4. La herramienta divide a las transformaciones de código fuente en dos tipos. Aquellas transformaciones

*implementada

que se realizan sobre el editor de texto activo y aquellas transformaciones que se realizan sobre un conjunto de archivos de un determinado proyecto. Cualquier transformación que quiera realizarse se debe implementar como una clase java que debe heredar de alguna de las dos super-clases según el tipo de transformación. Estas transformaciones son implementadas por Photran como refactorizaciones, a partir de este punto referirse a transformación de código fuente y refactorización será equivalente.

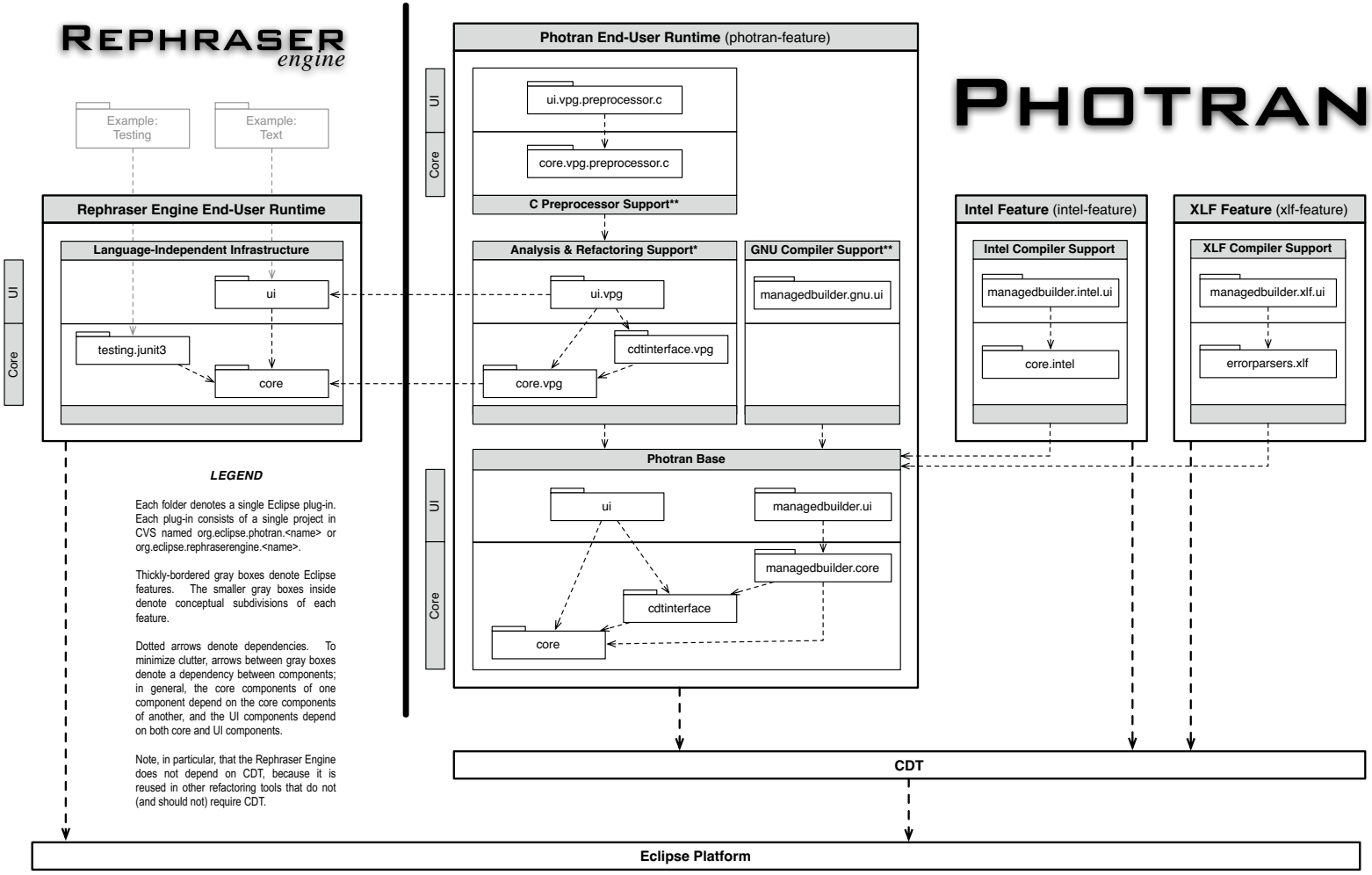


Figura 5.4: Photran, Fortran View

5.2.1. Ejemplo Uno: Modernización de las Instrucciones DO

Una de las curiosas características de Fortran, como ya se ha especificado anteriormente, es la posibilidad de escribir de distintas formas la misma funcionalidad. El ejemplo más interesante desde el punto de vista del Cómputo Científico, ya que gran parte del mismo se basa en ella, es la instrucción DO. A continuación pueden apreciarse las tres formas de escribir un DO en la actualidad en Fortran:

```
....                        ....
    DO 110 I=1,10               DO 100 I=1,10
    DO 100 J=1,10               DO 100 J=1,10
    MATRIX(I,J)=0           100 MATRIX(I,J)=0
100 CONTINUE                ....
110 CONTINUE                ....
...                         ....
      (A)                         (B)

....
DO I=1,10
   DO J=1,10
     MATRIX(I,J)=0
   END DO
END DO
....
      (C)
```

Las formas (A) y (B) son las que inicialmente utilizaban etiquetas en el código fuente para realizar el ciclo, pues la introducción de la instrucción END DO fue hecha en el estándar de Fortran 90 [9]. Por ende, todo código escrito con anterioridad a la publicación del estándar de Fortran 90 utilizaba etiqueta o la instrucción CONTINUE para finalizar el bloque del DO. La forma (B) de escritura del ciclo es llamada SHARED DO LOOP, pues ambas instrucciones DO comparten el mismo final, actualmente marcada como obsoleta en el Apéndice B del Estándar de Fotran [116]. Una de las transformaciones implementadas es aquella capaz de modificar la forma antigua de escribir los ciclos, por decirlo de alguna manera, a

Code Before

```
    program main

     ! Shared Do
     ! Loop Termination
     do 100 j=1,10
     do 100 w=1,10
100  i=j+1


end program main
```

Code After

```
  program main

    ! Shared Do
    ! Loop Termination
    do j=1,10
       do w=1,10
            100 i=j+1
       end do
    end do

end program main
```

Figura 5.5: Transformación de un Shared DO loop

la forma moderna en la cual finalizan con la instrucción END DO. A continuación se muestra un ejemplo del comportamiento de dicha transformación en el caso de los SHARED DO LOOPS, como puede apreciarse en la Figura 5.5.

En el caso de los ciclos terminados en una etiqueta con una instrucción CONTINUE, el comportamiento de la transformación debería ser la que se muestra en la Figura 5.6.

Es importante determinar cuáles son la precondiciones necesarias para la transformación. Las precondiciones deben garantizar tres cosas [172]. La primera es que la entrada del usuario sea válida. La segunda es garantizar que luego de que la transformación sea realizada, el código fuente generado compile. Y por último, si la transformación es realizada, el código resultante debe tener el mismo comportamiento que el programa original. Según la complejidad de la transformación la cantidad y complejidad de las precondiciones variará. Las precondiciones de la transformación en cuestión son:

- El código fuente debe tener por lo menos una instrucción DO.
- La etiqueta en la que termina la instrucción DO debe ser única.
- La instrucción de terminación del DO debe estar en el mismo nivel de anidamiento que la instrucción DO.

Code Before

```
program main

  ! simple Do Loop
  do 110 i = 1,10
110 j=i

! simple Do Loop2
    do 120 i = 1,10
      USAV(I,K)=UCLIN(I,K)
      VSAV(I,K)=VCLIN(I,K)
      UCLIN(I,K)=UP(I,K)
      VCLIN(I,K)=VP(I,K)
120 continue


end program main
```

Code After

```
  program main

! simple Do Loop
do i = 1,10
    110 j=i
END DO

! simple Do Loop2
do i = 1,10
    USAV(I,K)=UCLIN(I,K)
    VSAV(I,K)=VCLIN(I,K)
    UCLIN(I,K)=UP(I,K)
    VCLIN(I,K)=VP(I,K)
END DO

end program main
```

Figura 5.6: Transformación de Diversos Tipos de Ciclos DO Escritos en Formato Antiguo

Una vez definidas las precondiciones deberán implementarse en la subclase java correspondiente. Posteriormente se deberá realizar la transformación del código fuente mediante la modificación del AST. En la Figura 5.7 se muestra el código fuente original y posteriormente, una vez seleccionada la transformación que debe realizarse, se visualiza una vista que muestra el antes y el después de cómo quedaría el código transformado Figura 5.8 y por último puede verse en la Figura 5.9 el código fuente transformado.

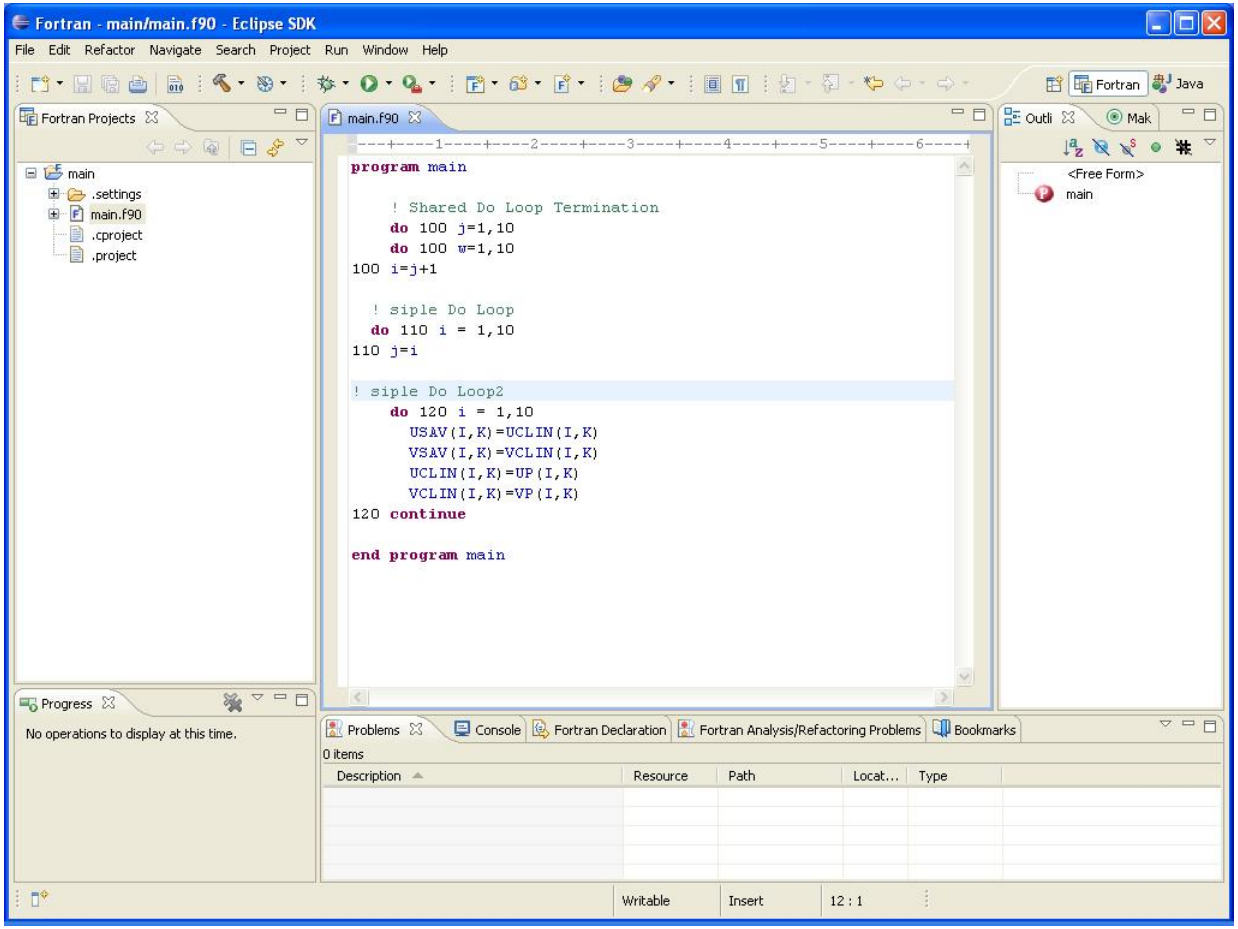


Figura 5.7: Transformación de Modernización de Ciclos DO Paso 1.

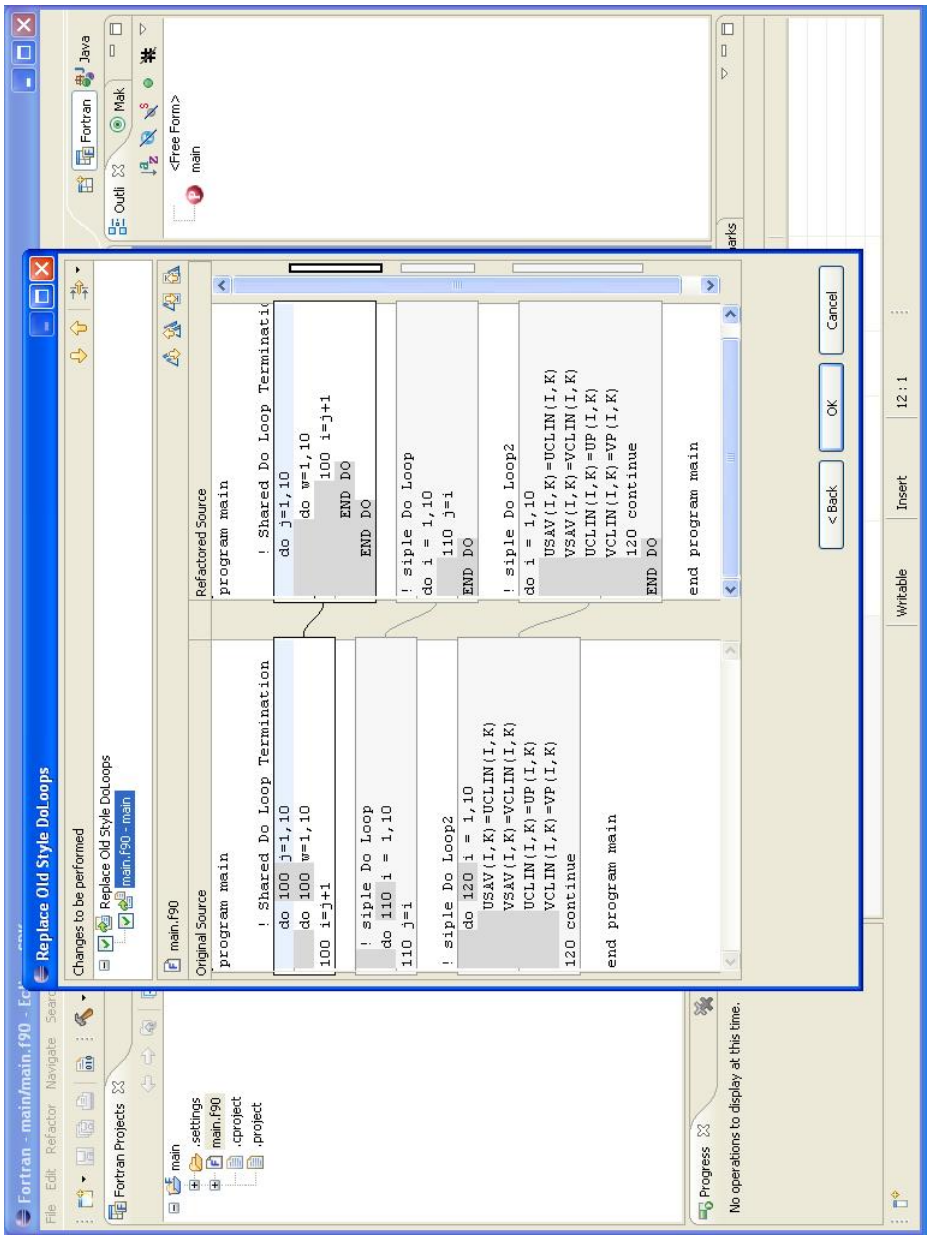


Figura 5.8: Transformación de Modernización de Ciclos DO Paso 2.

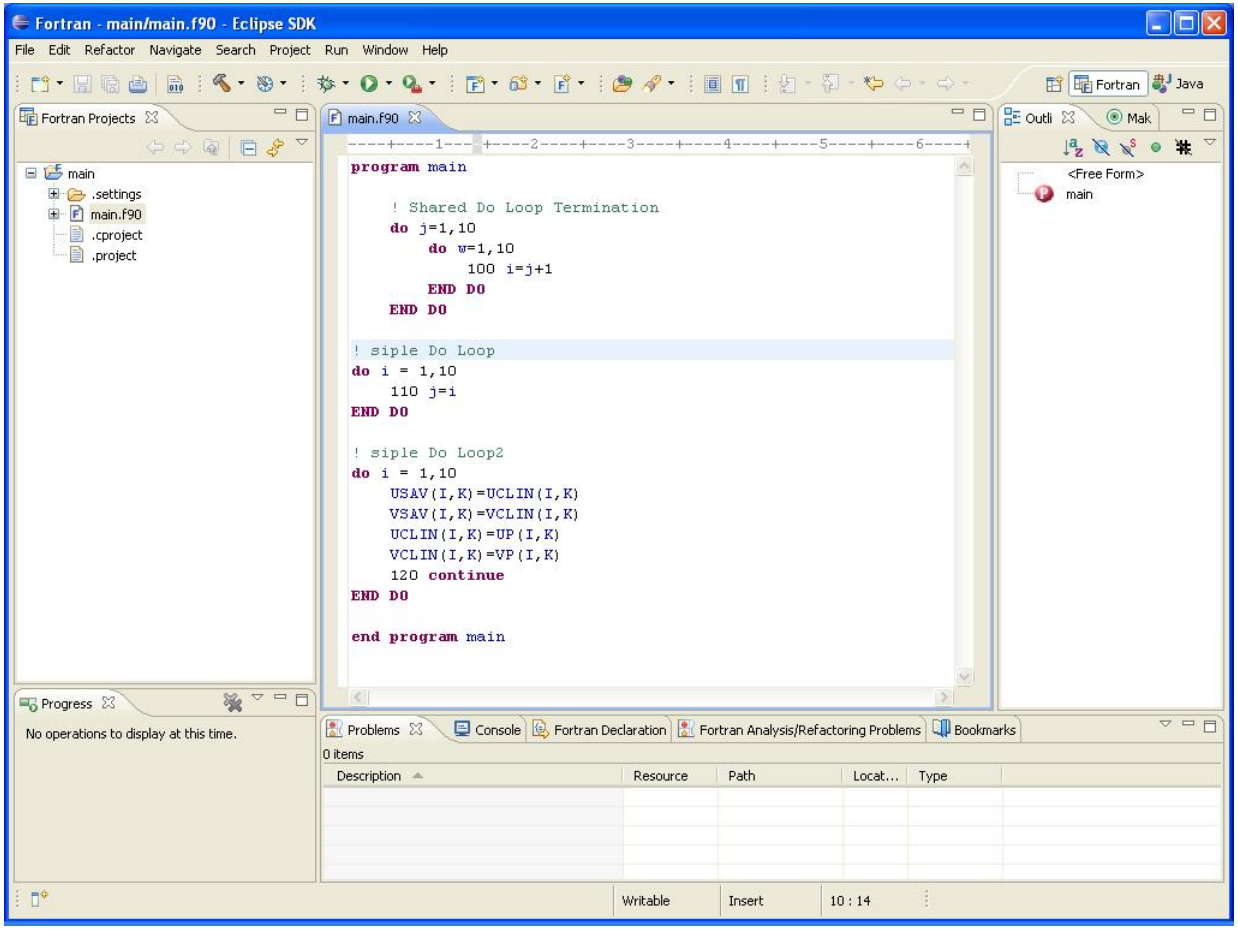


Figura 5.9: Transformación de Modernización de Ciclos DO Finalizada.

5.2.2. Ejemplo Dos: Pasar de FORTRAN77 a Fortran 90 con Formato Libre

Hasta la versión del estándar de Fortran que dio lugar a Fortran 90, todas las versiones anteriores del lenguaje sólo aceptaban como entrada código fuente en formato fijo. En la sección 3.3 del estándar de Fortran 90 [9] se definen dos tipos de formatos de código fuente. En el apartado 3.3.1 de dicho estándar se describe al formato libre en el cual se define como caracter de comentario al "!" y como caracter de continuación de línea a "&". Por otro lado en el apartado 3.3.2 del estándar se define al formato fijo cuyas líneas tienen una longitud de 72 caracteres, las instrucciones deben comenzar en la columna 7, los comentarios en la columna 1 con los caracteres "C" , "*" o "!". Además, cualquier caracter en la posición 6 define la continuación de línea. Dado que los programadores modernos prefieren el formato libre de escritura al formato fijo ya que es más flexible a la hora de la edición se ha implementado una transformación que se encarga de cambiar de formato fijo a formato libre. La transformación se muestra a continuación, en la Figura 5.10.

Las precondiciones para esta transformación son simples pues se requiere que el código fuente del programa o rutina seleccionado esté escrito en formato fijo. Esta transformación convierte los comentarios de formato fijo a formato libre, así como los caracteres de continuación de línea.

Específicamente para esta transformación se ha encontrado un bug en uno de los componentes del parser de Photran, más precisamente en la clase java llamada FixedFormLexerPrepass que se encarga de eliminar los espacios que pueden aparecer en el formato fijo, que en algunos casos no puede gestionar bien los comentarios del formato fijo, especialmente los que comienzan con "!".

Esta transformación se implementa en base a la posibilidad que tiene cada Token que forma parte del AST de almacenar los comentarios y los espacios en blanco del código fuente original [171]. A partir de esta propiedad se recorren los Tokens mediante la implementación de una clase que extiende un patrón visitante para recorrer los Tokens del árbol y para cada uno de los Tokens se obtienen los

comentarios haciendo los cambios pertinentes así como también se modifican los caracteres de continuación de línea.

5.2.3. Ejemplo Tres: Agregar el INTENT IN a los Parámetros de una Rutina

En el apartado 5.1.2.3 del estándar de Fortran 90 [85] se describen las características del atributo INTENT. Este atributo puede tener 3 especificadores de intent: IN, OUT, INOUT. Estos especificadores determinan si un argumento será exclusivamente leído (INTENT IN), será exclusivamente escrito (INTENT OUT) o si será leído y escrito dentro del ámbito del subprograma donde el parámetro está pasado como tal (INTENT INOUT).

La dinámica de la transformación es la siguiente:

1. En primer lugar el programador deberá estar editando con la herramienta un programa Fortran y tendrá seleccionado un parámetro formal de un subprograma.

2. La herramienta determinará si el parámetro formal varía debido a una asignación dentro del subprograma, de ser así no permitirá continuar con la transformación.

3. Si el parámetro formal es utilizado sólo para lectura de su contenido dentro del cuerpo del subprograma, entonces a la declaración del parámetro formal se le agregará el atributo INTENT con el especificador IN.

En la Figura 5.11 se muestra un ejemplo de dicha transformación. Para proceder con la transformación en primer lugar debemos determinar el conjunto de pre-condiciones a ser verificadas:

1. El texto seleccionado por el programador debe ser un parámetro formal definido en el subprograma.

2. El parámetro formal seleccionado no bebe tener definido su INTENT.

Code Before

```
|---5--7--1----------2-------
      program main

C simple Do Loop
      do 110 i = 1,10
      110 j=i
C simple Do Loop2
      do 120 i = 1,10
      USAV(I,K)=UCLIN(I,K)+
     &VSAV(I,K)-VCLIN(I,K)
      UCLIN(I,K)=UP(I,K)
      VCLIN(I,K)=VP(I,K)
  120 continue
      i=1
      if (i.lt.10) then
      i=1
      else
      i=1+1
      end if


      end program main
```

Code After

```
program main

! simple Do Loop
do 110 i = 1,10
    110 j=i
! simple Do Loop2
do 120 i = 1,10
    USAV(I,K)=UCLIN(I,K)+ &
    VSAV(I,K)-VCLIN(I,K)

    UCLIN(I,K)=UP(I,K)
    VCLIN(I,K)=VP(I,K)
120 continue
    i=1
    if (i.lt.10) then
       i=1
    else
       i=1+1
    end if


end program main
```

Figura 5.10: Transformación de Formato Fijo a Formato Libre

Code Before Refactoring

```
FUNCTION Square(r)

IMPLICIT NONE
REAL :: Square
REAL :: r

Square =  r * r

END FUNCTION Square
```

Code After Refactoring

```
FUNCTION Square(r)

IMPLICIT NONE
REAL :: Square
REAL :: r
INTENT(IN) :: r

Square =  r * r

END FUNCTION Square
```

Figura 5.11: Transformación Agregar el Especificador INTENT

3. El parámetro formal seleccionado no debe ser asignado dentro de todo el cuerpo del subprograma.

4. En el ámbito del subprograma no puede haber definiciones implícitas, para ello se debe utilizar la instrucción IMPLICIT NONE.

5. El subprograma tiene que ser una rutina interna.

La pre-condición 1 valida la entrada seleccionada por el programador, en este caso que el texto seleccionado sea un parámetro formal definido en el subprograma. A su vez, las pre-condiciones 2 y 3 se hacen cargo de que el programa compile, ya que realizan el control que un compilador Fortran haría. En este caso garantizar que el comportamiento del programa no varíe resulta trivial, pues la transformación no actúa sobre una instrucción ejecutable; además el atributo `INTENT(IN)` no cambia la semántica de ninguna otra instrucción ejecutable, por ende esta transformación no puede cambiar el comportamiento en tiempo de ejecución del programa.

Las pre-condiciones 4 y 5 no son requeridas para realizar la transformación, pero han sido agregadas para simplificarla. La pre-condición 4 garantiza que exista una declaración explícita para el parámetro formal. Por otro lado la pre-condición 5 garantiza que exista una única declaración de la rutina, si la misma fuera externa sería mucho más compleja la transformación.

5.3. Transformaciones Orientadas a la Paralelización

Esta forma de enfocar las transformaciones de código fuente pueden ser utilizadas no solamente para la modernización o actualización del mismo, sino que también es posible utilizar este enfoque para generar transformaciones que sin alterar la semántica de un programa secuencial permita de forma automatizada generar una versión paralela del mismo. De esta forma se podría pensar en implementar una transformación de código fuente destinada a la paralelización del mismo usando por ejemplo OpenMp [4] un estándar casi "de facto" para la paralelización en una arquitectura multi-core con memoria compartida, ver ejemplo en Figura 5.12.

```
a) Legacy

   DO 212  I = 1,MQ
         DPR(I, 1) =  ABS( PR(I, 2) - PR(I,  1) )  /  RO(I, 1)
         DPR(I,NQ) =  ABS( PR(I,NQ) - PR(I,NQ1) )  /  RO(I,NQ)
  212 CONTINUE

b) Parallelized

!$OMP PARALLEL DO PRIVATE (I)
!$OMP&SHARED(MQ, DPR, PR, RO )
   DO 212  I = 1,MQ
         DPR(I, 1) =  ABS( PR(I, 2) - PR(I,  1) )  /  RO(I, 1)
         DPR(I,NQ) =  ABS( PR(I,NQ) - PR(I,NQ1) )  /  RO(I,NQ)
  212 CONTINUE
```

Figura 5.12: Ciclo DO Perteneciente a Código Heredado Paralelizado Mediante Directivas OpenMP

Teniendo en cuenta la observación teórica que afirma que los cálculos independientes pueden ser ejecutados simultáneamente [107], el proceso de paralelización puede pensarse como trivial, muy lejos de ser cierto en la realidad. Un proceso automatizado de paralelización debe, como mínimo, verificar las condiciones de Bernstein [27]. Estas condiciones parten de P_i y P_j dos segmentos del programa. Sean I_i todas las variables de entrada y O_i todas las variables de salida de P_i y del mismo modo para P_j. Entonces P_i y P_j son independientes si satisfacen:

$$I_j \cap O_i = \varnothing,$$
$$I_i \cap O_j = \varnothing,$$
$$O_i \cap O_j = \varnothing.$$

Estas condiciones representan dependencias de flujo, anti-dependencia y dependencia de salida. El chequeo automatizado de este tipo de condiciones es muy complejo de implementar en la actualidad.

5.4. Resumen

A lo largo de este capítulo se ha descrito la forma de realizar una transformación a programas escritos en Fortran partiendo de su código fuente, utilizando un enfoque similar al que utilizan los compiladores, mediante la construcción de un AST y de su recorrido para chequear las pre-condiciones de la transformación, la reescritura y modificación del mismo AST. Se han planteado algunos ejemplos implementados, e integrados a una herramienta que está siendo utilizada en la actualidad por desarrolladores para la implementación de programas Fortran. Por otro lado se ha planteado, descrito e implementado la posibilidad de la construcción de transformaciones orientadas a la paralelización automática de programas secuenciales.

Capítulo 6

La Verificación: Los Resultados

Tanto dentro del proceso de Desarrollo de Software Guiado por el Cambio así como dentro de la etapa de mantenimiento del software es de fundamental importancia que los cambios realizados al software proporcionen el comportamiento deseado del mismo. En el ámbito de las aplicaciones científicas se debe tener en cuenta, además, que los resultados que estos programas arrojan tengan validez numérica dentro del dominio de aplicación.

6.1. La Verificación

Dentro del ámbito de la Ingeniería de Software el concepto de verificación está descrito como " *El proceso de evaluar a un sistema o componente para determinar si los productos de una determinada fase de desarrollo satisfacen las condiciones impuestas al inicio de esa fase*" [180]. A su vez según el Capability Maturity Model Integration también se define verificación como "*el proceso que asegura que un conjunto determinado de productos converjan a su especificación de requerimientos*" [198], en otras palabras la verificación desde la perspectiva de la ingeniería de software debe responder a la pregunta ¿Se está construyendo el producto correctamente?. Por otro lado, desde el punto de vista de la Computación Científica o de la Computational Science and Engineering (CSE) el proceso de verificación, además de cumplir con las definiciones anteriores, también debe pensarse como

"la evaluación de la precisión de la solución del modelo computacional" [167]. Esta definición presenta la idea de que además de determinar si se está implementando el programa correcto también hay que tener en cuenta la perspectiva de la Computación Científica si los resultados del mismo también son los correctos. Por ejemplo si la precisión de los cálculos es acorde a los necesitados, si el error de cálculo se ajusta a lo esperado, entre otras verificaciones. Por otro lado desde el ángulo de HPC la verificación además de incluir los dos puntos de vista anteriormente mencionados debe centrarse en los tiempos de respuesta de un determinado software en un determinado hardware.

Teniendo en cuenta el amplio abanico temático que abarca el concepto de verificación dentro del ámbito de la producción de software, que por sí mismo constituye un trabajo de investigación completo, este capítulo se centrará en la etapa de verificación desde el enfoque de las transformaciones orientadas a la paralelización.

Para ello se ha hecho especial hincapié en la forma de verificar que los cambios realizados al programa efectivamente optimicen el funcionamiento del programa respecto al tiempo de cómputo del mismo. En este caso querrá verificarse que ciertos parámetros que tienen que ver con la ejecución del programa hayan variado respecto de la versión original. Para ello se ha buscado integrar al entorno de desarrollo con la habilidad para chequear características del hardware en tiempo de ejecución del software en forma automatizada. Haciendo así que el programador se enfoque en "Qué" verificar y no en "Cómo" hacerlo.

Es posible remarcar tres facetas destacables de la verificación del software relacionada con los sistemas heredados de cómputo intensivo escritos en Fortran, éstas son:

- La verificación de la funcionalidad: que consiste en determinar si la funcionalidad del software no ha variado tras el proceso de transformación.

- La verificación numérica: que consiste en determinar si los resultados numéricos no han variado tras la aplicación de los cambios o transformaciones.

- La verificación del rendimiento: que consiste en determinar si los resultados a nivel cantidad de cómputo por unidad de tiempo han variado o no tras la aplicación de los cambios o transformaciones.

El abanico de posibilidades y herramientas que abren estos tres enfoques, respecto al proceso de verificación es realmente enorme quedando fuera del alcance del presente trabajo. Dado que el foco de este trabajo no se centra en la etapa de verificación, se ha realizado un pequeño aporte que además funciona como prueba de concepto.

6.1.1. Los Contadores de Hardware

Los contadores de hardware aparecen como un efecto de una técnica de diseño de hardware conocida como DFT (Design For Test) o Diseño para testear [209]. Esta técnica fue utilizada desde la década de los 40 por los ingenieros electrónicos para chequear algunas características de computadoras análogas como por ejemplo el voltaje. En nuestros días esta técnica de medición es utilizada en los micro-procesadores modernos en lo que se conoce como contadores de hardware. Los contadores de hardware tienen una historia por demás interesante, el caso más famoso es de los procesadores Pentium de Intel. Hacia 1994 un conjunto de registros no documentados del procesador Pentium fueron encontrados al de-compilar código fuente [190, 145]. El fabricante no había dado a conocer estos registros especiales que figuraban como confidenciales y propietarios. En la actualidad la política de Intel ha cambiado respecto a estos registros dando una completa especificación de los mismos [113, 114]. En la cuadro 6.1 se puede apreciar la descripción de la cantidad de contadores para las distas arquitecturas de procesadores de Intel

A partir de la salida a la luz de la capacidad de medir el comportamiento del hardware o más precisamente del microprocesador, los programadores han utilizado esta capacidad para medir y analizar el funcionamiento de sus programas en las distintas plataformas existentes. Hay dos formas diferentes de realizar dicho análisis. La primera y la más común es mediante la utilización de software cons-

truido por terceros como por ejemplo: perf, perfsuite, perfmon, Oprofile, Vtune, etc. La segunda y menos conocida forma de llevar a cabo dicho análisis (tal vez por su dificultad) es mediante la utilización de bibliotecas que permiten realizar las mediciones directamente en el código fuente del programa a través de una API como por ejemplo: la biblioteca PAPI, Rabbit, PCL, entre otras. Tal vez esta última forma de realizar las mediciones utilizando los contadores de hardware no ha tenido mucha uso debido a su compleja implementación. Pero si se analiza detenidamente el proceso de medición se puede extraer un patrón de utilización de las mismas. Este patrón puede ser dividido en pasos o fases:

1. Inicializar la biblioteca.
2. Crear el conjunto de eventos a medir.
3. Agregar los eventos a ser medidos (eje. cantidad de accesos a Memoria Cache).
4. Comenzar la medición.
5. Parar la medición.
6. Mostrar los resultados.

Dado que todos estos pasos deben ser realizados por el programador dentro del código fuente del programa muchas veces se vuelve tedioso e incluso puede

Processor Family	Hardware counter
P5 Pentium Processors	2
P6 Family Processor	2
Intel® CoreTM Microarchitecture	3 f.c.p.t
Procesadores Basados en Intel NetBurst® Microarch.	18 g.p.p.c.
Procesadores Basados en Intel Sandy Bridge Microarch.	8 g.p.p.c. + 3 f.c.p.t
Procesadores Basados en Intel Westmere Microarch.	8 g.p.p.c. + 3 f.c.p.t
Procesadores Basados en Intel Nehalem Microarch.	4 g.p.p.c. + 3 f.f.p.c
Intel® AtomTM Microarchitecture	2 g.p.p.c. + 3 f.f.p.c

Cuadro 6.1: Contadores de Hardware Según la Micro-arquitectura de los Procesadores Intel.
g.p.p.c=general-purpose performance counters. f.f.p.c=fixed-function performance counters. f.c.p.t=function counter per thread.

llegar a contribuir a la aparición de errores en el mismo. Por ello automatizar este proceso de medición ayudaría a los programadores a despreocuparse de la implementación y ocuparse de la medición.

6.2. La Biblioteca PAPI

PAPI es la abreviatura de Performance Application Programming Interface, esta biblioteca permite realizar la medición de un conjunto de eventos que forman parte de los eventos medibles del microprocesador según la arquitectura [142, 164, 39]. En la cuadro 6.2 se muestra la interfaz que presenta la biblioteca PAPI para Fortran.

Subroutine	Description
PAPIF_accum	Accumulate and reset counters in an event set.
PAPIF_add_event	Add PAPI preset or native hardware event to an event set.
PAPIF_add_named_event	Add PAPI preset or native hardware event to an event set by name.
PAPIF_add_events	Add multiple PAPI presets or native hardware events to an event set.
PAPIF_cleanup_eventset	Empty and destroy an EventSet.
PAPIF_create_eventset	Create a new empty PAPI EventSet.
PAPIF_assign_eventset_component	Assign a component index to an existing but empty EventSet.
PAPIF_destroy_eventset	empty and destroy an EventSet.
PAPIF_get_dmem_info	Get information about the dynamic memory usage of the current program.
PAPIF_get_exe_info	Get information about the dynamic memory usage of the current program.
PAPIF_get_hardware_info	Get information about the system hardware .
PAPIF_num_hwctrs	Return the number of hardware counte1rs on the cpu.

Cuadro 6.2: FORTRAN PAPI API

Existen varios niveles de complejidad en la utilización de dichas bibliotecas. El primero es la posible introducción de errores adicionales en el código fuente por la utilización de estas bibliotecas. La segunda es la variación de los eventos de los contadores según la plataforma en que se esté realizando el análisis. Finalmente la impresión de los resultados también agrega complejidad a dicha tarea. Por lo tanto se ha implementado una herramienta que integra en forma automatizada, en primer lugar la captura de los eventos disponibles en la plataforma de trabajo, en segundo lugar la selección de los eventos a medir y en tercer y último lugar la generación automática del código fuente necesario para realizar la medición e impresión de los resultados obtenidos de la medición. Esta herramienta a su vez se ha integrado al entorno de programación Photran. Cabe destacar que de la misma forma que existe una transformación para agregar el código fuente para realizar

la medición, existe otra que elimina el código fuente agregado para utilizar los contadores de hardware.

6.2.1. Integración de los Contadores de Hardware

La herramienta para integrar los contadores de hardware consta de dos partes. La primera, interactuar con la instalación de la biblioteca PAPI para:

1. Determinar la versión de la biblioteca.
2. Mostrar dicha versión al usuario o informar que dicha biblioteca no está instalada.
3. Obtener la lista de eventos medibles de la plataforma.
4. Mostrar la lista de los eventos medibles en la plataforma.

Para ello se implementaron las interfaces de usuario necesarias para realizar dicha funcionalidad a través de los puntos de extensión que proporciona Eclipse para contribuir con nueva funcionalidad [93, 52], ver Figuras 6.1.

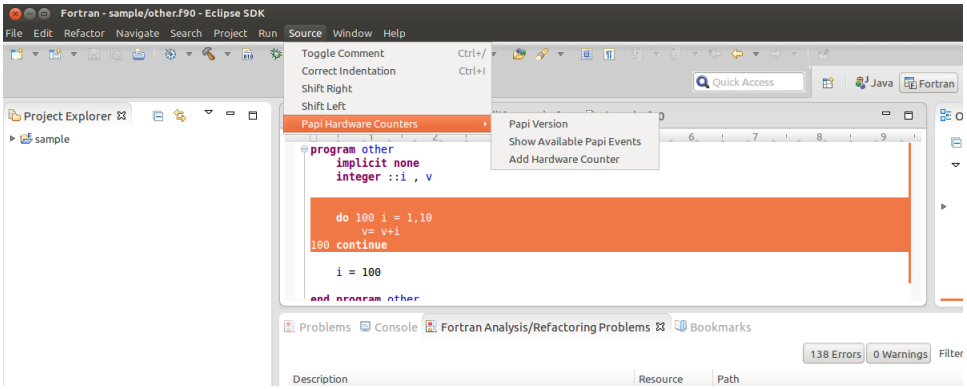


Figura 6.1: Las Dos Contribuciones en el Menú

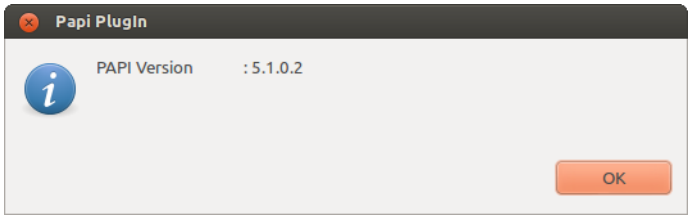


Figura 6.2: Diálogo que Muestra la Versión de la Biblioteca PAPI

La segunda parte de la herramienta obtiene los eventos de hardware medibles y le proporciona al usuario una lista de dichos eventos para que el programador seleccione y a partir de dicha selección los agregue en la porción de código fuente que éste quiera analizar, ver Figura 6.4.

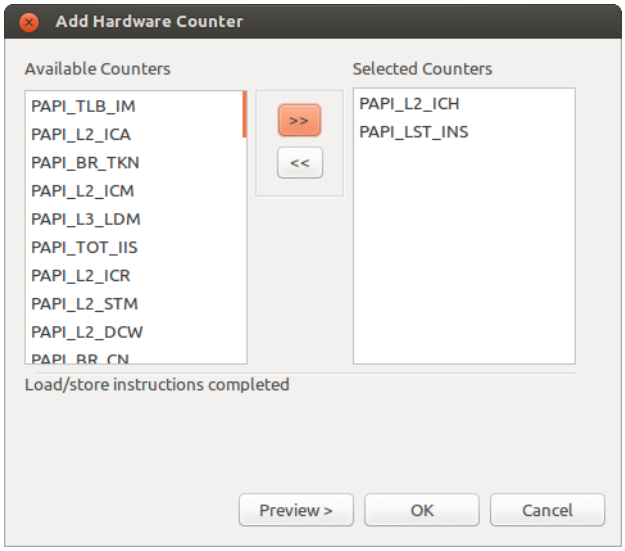


Figura 6.3: Diálogo que Permite Seleccionar los Ventos a Medir.

Al finalizar la selección se podrá previsualizar cómo quedará el código fuente mediante un diálogo que muestra el antes y el después de la transformación del código fuente.

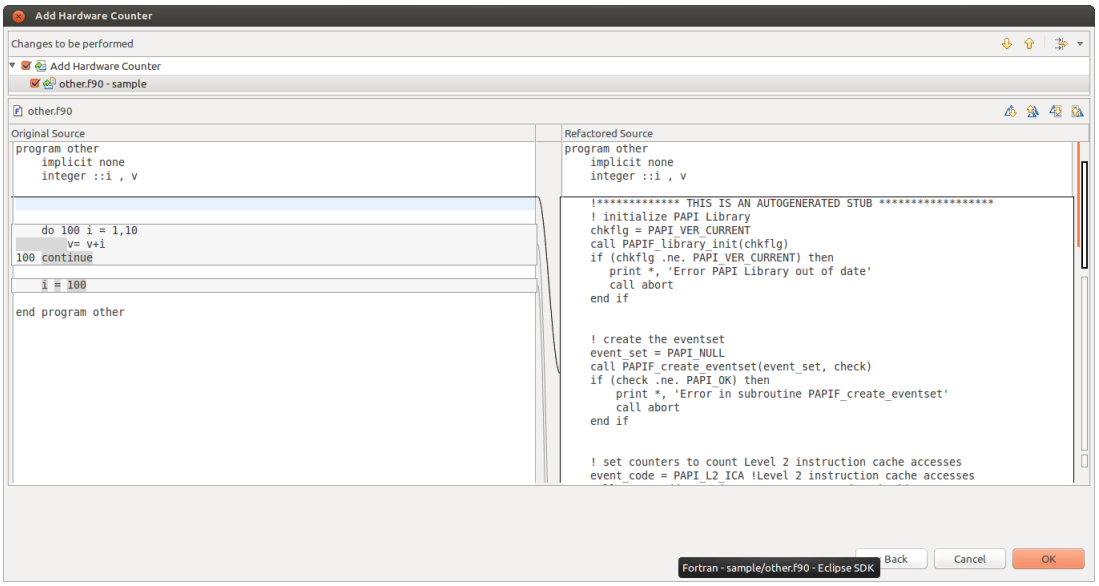


Figura 6.4: Vista de Diferencias Antes de Aplicar los Cambios.

6.2.2. La transformación

Éste es el componente más complejo de la herramienta de integración de los contadores de hardware ya que debe agregar automáticamente código fuente al programa existente y asegurarse que el mismo compile. Para ello una vez más se utilizará el AST del programa para realizar dichos cambios, pues con el código generado mediante el AST se asegurará la correcta inclusión del mismo ya que éste debe pasar por el parser con la consiguiente verificación sintáctica, antes de ser agregado al AST. Una vez que el usuario, en este caso el programador, selecciona los eventos a medir la herramienta deberá generar el código fuente necesario para: inicializar la biblioteca, crear el conjunto de eventos a medir, agregar los eventos a ser medidos, comenzar la medición, finalizar la medición y luego mostrar los resultados. Todo esos pasos deben envolver el código a ser analizado. Por ello la inicialización, la adición de eventos a medir y el inicio de la medición deberán ser agregados antes de la primera instrucción del bloque de código seleccionado:

1. Inicialización:

```
! initialize PAPI Library
```

```
chkflg = PAPI_VER_CURRENT
call PAPIF_library_init(chkflg)
if (chkflg .ne. PAPI_VER_CURRENT) then
   print *, 'Error PAPI Library out of date'
   call abort
end if
```

2. Creación del conjunto de eventos:

```
! create the eventset
event_set = PAPI_NULL
call PAPIF_create_eventset(event_set, check)
if (check .ne. PAPI_OK) then
   print *, 'Error in subroutine PAPIF_create_eventset'
   call abort
end if
```

3. Adición al conjunto de eventos de los eventos seleccionados:

```
! set counters to <should be replaced by the event escription>
event_code = PAPI_L2_ICH
call PAPIF_add_event(event_set, event_code, check)
if (check .NE. PAPI_OK) then
   print *, "Abort After PAPIF_add_events: ", check
   call abort
endif
```

4. Iniciar el análisis:

```
! start counting loads/stores
call PAPIF_start(event_set, check)
if(check .ne. PAPI_OK) then
   print *, 'Abort after PAPIF_start: ', check
```

```
   call abort
endif
```

A partir de la última instrucción del bloque de código seleccionado se deberá generar el código fuente que finalice el análisis y aquella parte del código fuente que muestre los resultados del mismo.

1. Finalizar el análisis:

```
! stop counting
call PAPIF_stop(event_set, values, check)
if (check .ne. PAPI_OK) then
   print *, 'Abort after PAPIF_stop: ', check
   call abort
endif
```

2. Mostrar los resultados:

```
print *, 'Number of <should be replaced by the event name>:  ', values(1)
print *, 'Number of <should be replaced by the event name>:  ', values(2)
   ...      ...      ...      ...
print *, 'Number of <should be replaced by the event name>:  ', values(<N>)
```

A continuación se muestra un ejemplo de la utilización de la herramienta descrita anteriormente. A partir del clásico ejemplo de multiplicación de matrices al cual se le realizará un análisis de performance utilizando este proceso:

```
program matmul

 integer             :: row,col
 integer             :: i,j,k
 integer,allocatable  :: a(:,:)
 integer,allocatable  :: b(:,:)
 integer,allocatable  :: c(:,:)

 row=3000
```

```
col=3000

allocate(a(1:row,1:col))
allocate(b(1:row,1:col))
allocate(c(1:row,1:col))

 ! init matrices a and b such that c(i, j) = n
 do i = 1,row
   do j=  1,col
     a(i,j) = 1
     b(i,j) = 1
     c(i,j) = 0
   end do
 end do



 do i=1,row
   do j=1,col
    do k=1,col
         c(i, j) =c(i, j) + a(i, k) * b(k, j)
     end do
   end do
 end do



! Just checking...
 DO i = 1, col
   DO j = 1, row
     IF (.not. (c(i, j) .eq. col)) THEN
       PRINT *, "Error on value of c(", i, ", ", j, ") ", c(i, j)
     END IF
   END DO
 END DO
 PRINT *, "No problems found at c if nothing reported..."

 deallocate(a)
 deallocate(b)
 deallocate(c)

end program  matmul
```

Para ello se puede ver el proceso completo integrado al entorno de desarrollo en las Figuras 6.5, 6.6, 6.7, 6.8 y 6.9.

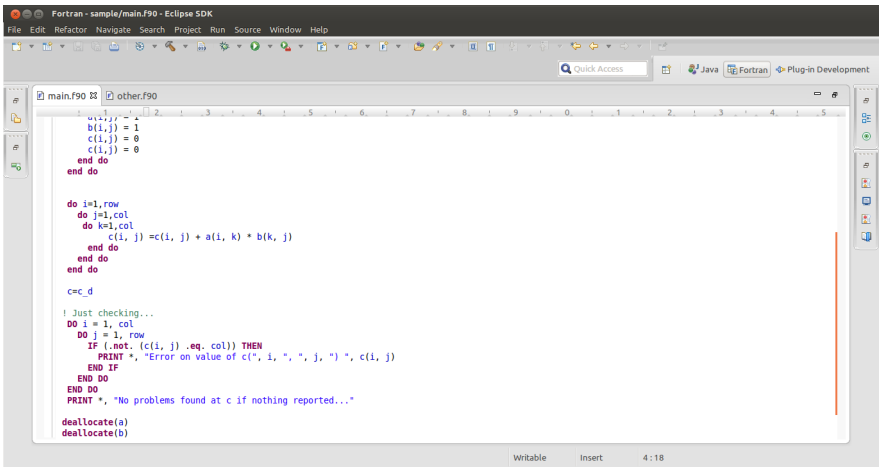


Figura 6.5: Código Fuente Inicial

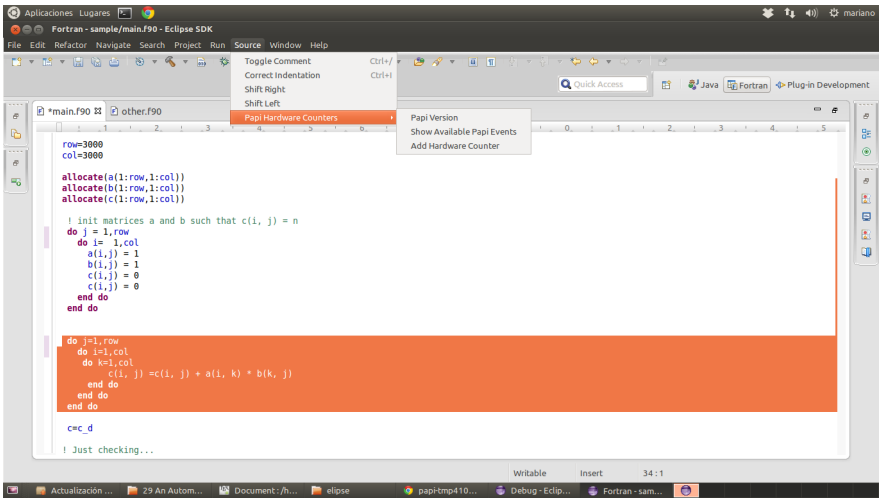


Figura 6.6: Código Fuente seleccionado a ser Evaluado

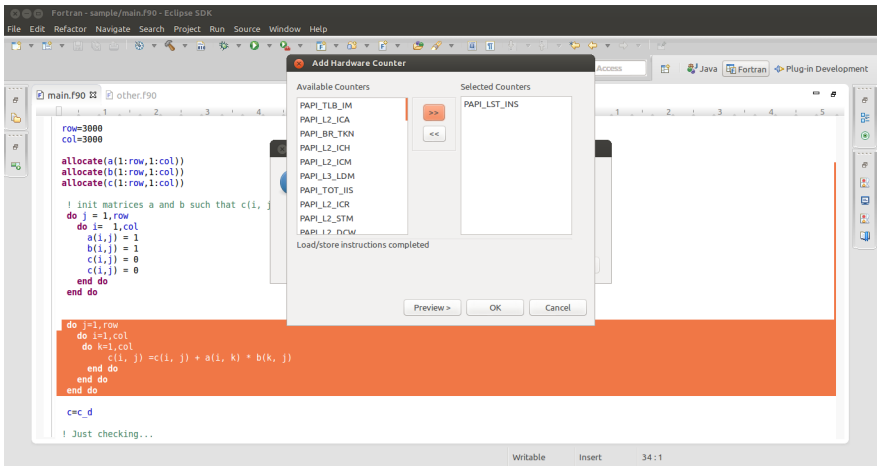


Figura 6.7: Selección de los Eventos de Hardware a Evaluar

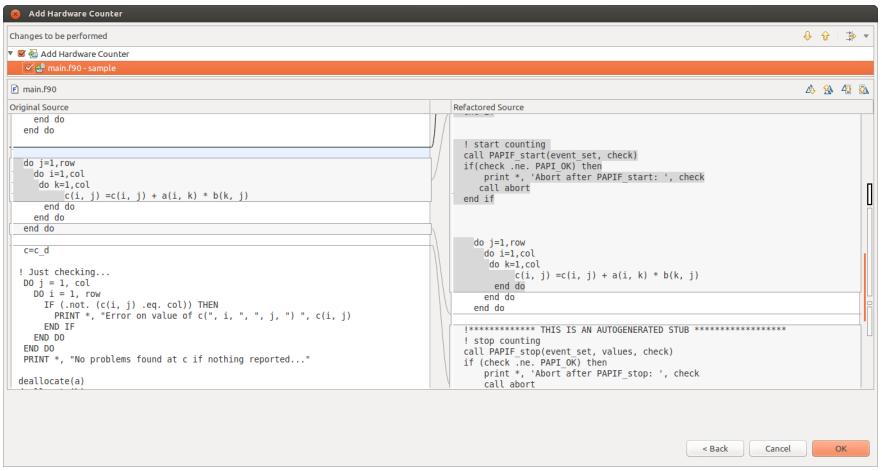


Figura 6.8: Vista de Diferencia Antes de Aplicar la Transformación

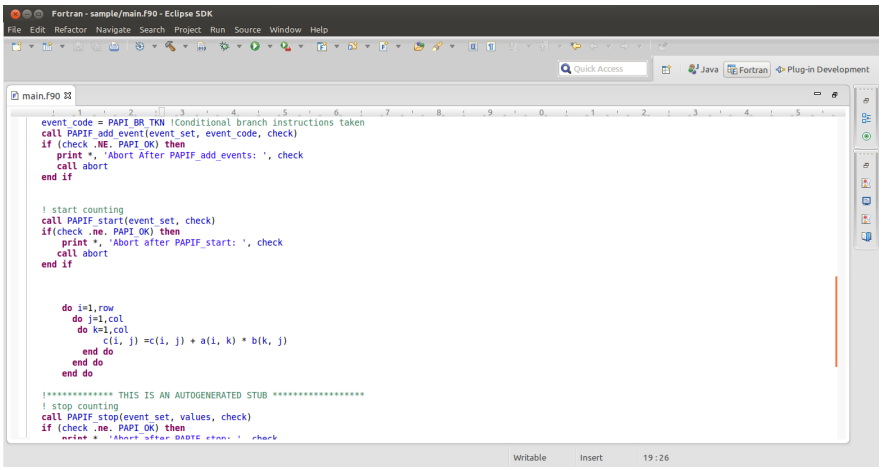


Figura 6.9: Código Fuente Transformado y Listo para Ser Evaluado

El código fuente Fortran resultante después de realizar la transformación es el siguiente:

```
program matmul
 implicit none
 integer, parameter  :: NUM_EVENTS =1
 integer, dimension(NUM_EVENTS)  :: event_set
 integer*8, dimension(NUM_EVENTS)    :: values
 integer :: check
 integer::event_code


 integer               :: row,col
 integer               :: i,j,k
 integer,allocatable :: a(:,:)
 integer,allocatable  :: b(:,:)
 integer,allocatable  :: c(:,:)

 row=100
 col=100

 allocate(a(1:row,1:col))
 allocate(b(1:row,1:col))
 allocate(c(1:row,1:col))

  ! init matrices a and b such that c(i, j) = n
  do i = 1,row
    do j=  1,col
```

```
    a(i,j) = 1
    b(i,j) = 1
    c(i,j) = 0
  end do
end do

!************* THIS IS AN AUTOGENERATED STUB ******************
! initialize PAPI Library
chkflg = PAPI_VER_CURRENT
call PAPIF_library_init(chkflg)
if (chkflg .ne. PAPI_VER_CURRENT) then
   print *, 'Error PAPI Library out of date'
   call abort
end if

! create the eventset
event_set = PAPI_NULL
call PAPIF_create_eventset(event_set, check)
if (check .ne. PAPI_OK) then
print *, 'Error in subroutine PAPIF_create_eventset'
call abort
end if

! set counters to count Conditional branch instructions taken
event_code = PAPI_BR_TKN !Conditional branch instructions taken
call PAPIF_add_event(event_set, event_code, check)
if (check .NE. PAPI_OK) then
   print *, 'Abort After PAPIF_add_events: ', check
   call abort
end if

! start counting
call PAPIF_start(event_set, check)
if(check .ne. PAPI_OK) then
print *, 'Abort after PAPIF_start: ', check
   call abort
end if

    do i=1,row
      do j=1,col
       do k=1,col
            c(i, j) =c(i, j) + a(i, k) * b(k, j)
        end do
```

```
        end do
      end do

  !************* THIS IS AN AUTOGENERATED STUB ******************
  ! stop counting
  call PAPIF_stop(event_set, values, check)
  if (check .ne. PAPI_OK) then
      print *, 'Abort after PAPIF_stop: ', check
      call abort
  end if

  print *, 'Number of Conditional branch instructions taken :  ', values(1)

 ! Just checking...
  DO i = 1, col
    DO j = 1, row
      IF (.not. (c(i, j) .eq. col)) THEN
        PRINT *, "Error on value of c(", i, ", ", j, ") ", c(i, j)
      END IF
    END DO
  END DO
  PRINT *, "No problems found at c if nothing reported..."

 deallocate(a)
 deallocate(b)
 deallocate(c)

end program  matmul
```

En este caso se ha medido el evento PAPI_BR_TKN que representa los saltos condicionales tomados. tras ser compilado, el mismo programa debería mostrar cuántas veces el evento de salto condicional ha sido utilizado en la ejecución de esa porción de código. Es importante destacar que en el proceso de paralelización es de vital importancia poder verificar el comportamiento del hardware.

6.3. Resumen

En este capítulo se ha presentado la fase de verificación del proceso de desarrollo propuesto. Dicha fase por sí misma es muy compleja y requiere de un estudio

más pormenorizado que escapa al alcance del presente trabajo. En el mismo se ha diseñado, implementado e integrado una herramienta que permite la inclusión automática del código fuente necesario para la utilización de la biblioteca de medición de eventos de hardware PAPI (Performance Application Programming Interface). A su vez se ha determinado que existe una serie de pasos específicos para la utilización de este tipo de biblioteca. Este conjunto de pasos la inicialización la biblioteca, la creación el conjunto de eventos a medir, la adición los eventos a ser medidos, el inicio de la medición, la finalización de la medición y la presentación de los resultados permiten la automatización de este proceso.

Capítulo 7

Aplicación del Proceso: Caso de Estudio 1

En este capítulo se estudiará la aplicación del proceso paso a paso. Para ello se mostrarán y describirán cada una de las fases y el flujo de trabajo propuesto en el los capítulos anteriores. Utilizando un ejemplo sobre código fuente Fortran extraído de bibliografía especializada en Computación Numérica.

7.1. Caso de Estudio 1: Introducción

En este primer caso de estudio utilizaremos un ejemplo extraído del libro **Numerical Mathematics and Computing** [47] escrito en FORTRAN 77. Lo interesante de este caso es que en su edición posterior del libro, del año 2003 [48], puede hacerse una comparación con la misma versión del programa en Fortran 90 cuyas modificaciones han sido realizadas por el autor del libro. La versión FORTRAN 77 del programa seleccionado evalúa la derivada de una función en un punto, el código fuente puede verse en la Figura 7.1.

```
      PROGRAM FIRST
C NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
C
C FILE: first.f
C
C FIRST PROGRAMMING EXPERIMENT
C
      DATA N/25/, H/1.0/, X/0.5/
      F = SIN(X)
      G = COS(X)
      DO 2 I = 1,N
      H = 0.25*H
      D = SIN(X + H) - F
      Q = D/H
      E = ABS(G - Q)
      PRINT *,H,D,Q,E
2     CONTINUE
      STOP
      END
```

Figura 7.1: Código Fuente del Programa FIRST.f

La idea básica de este caso es partir de una lista de cambios y determinar si la versión obtenida después de varias aplicaciones del proceso es parecida a la versión publicada en el libro de Fortran 90. La lista de cambios a aplicar al siguiente programa es:

1. Pasar a formato libre.
2. Cambiar a minúsculas las instrucciones del lenguaje.
3. Cambiar a minúsculas los nombres de los identificadores.
4. Introducir la instrucción IMPLICIT NONE.
5. Reemplazar los ciclos DO escritos en la forma antigua a la moderna.
6. Eliminar etiquetas no utilizadas.
7. Eliminar la instrucciones no requeridas(*).
8. Agregar a la instrucción END PROGRAM el nombre del programa.

Lista de Cambios

C	A	M	B	I	O	1
C	A	M	B	I	O	2
C	A	M	B	I	O	3
C	A	M	B	I	O	4
C	A	M	B	I	O	5
C	A	M	B	I	O	6
C	A	M	B	I	O	7
C	A	M	B	I	O	8
C	A	M	B	I	O	9
C	A	M	B	I	O	10

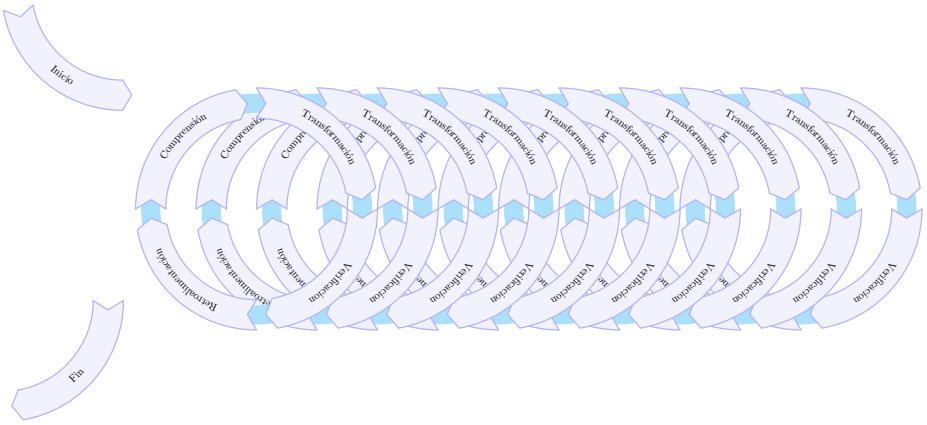


Figura 7.2: Proceso de Desarrollo Dirigido por el Cambio para FIRST.F77

9. Reemplazar el bloque DATA por las correspondientes variables(*).

10. Reemplazar el bloque DATA por las correspondientes constantes.

7.1.1. Aplicación del proceso

A partir de la lista de diez cambios definida anteriormente se planificará una iteración del proceso por cada cambio de la lista. En cada iteración se utilizará el flujo de trabajo propuesto en el Capítulo 3, ver Figura 7.2

Iteración 1: Pasar a formato libre

En esta primera iteración se establece ante todo la versión inicial del código fuente, para ello utilizaremos la herramienta git (`http://git-scm.com/`), un sistema de control de versiones, y el plug-in de Eclipse Egit `http://eclipse.org/egit/`. Una vez establecida la versión inicial se procede a obtener una salida de los resultados de esta versión, ver Figura 7.4, Figura 7.5 y Figura 7.6. Una vez realizados estos pasos se procede al commit de la versión considerada inicial

para la primera iteración con los resultados de la ejecución de la misma como un archivo del proyecto. Para ello utilizaremos el comando commit and push de Egit.

Tras realizar este proceso, se comienza con el cambio planteado. En este caso la transformación que se debe realizar tiene que ver con una característica que acompaña a Fortran desde sus comienzos [13, 86]. El formato fijo en Fortran consiste en una línea de código fuente escrito que debe satisfacer las siguientes reglas: de la columna 1 a la 5 se declaran las etiquetas, en la columna 6 se indicará las continuación de línea, de la columna 7 a la 72 se debe escribir código Fortran y de la línea 73 en adelante se escribirán comentarios opcionalmente [87, 84, 85, 116]. Esta restricción en la escritura de un programa Fortran que dura más de 36 años, hace que se encuentre código fuente difícil de leer y comprender. Un punto importante a tener en cuenta es que el formato fijo permite espacios ya sea en medio de una instrucción o de un nombre de una variable, por ejemplo "D O100I=1 ,10" en lugar de "DO 100 I = 1, 10". Esta propiedad del lenguaje tuvo relevancia, por ejemplo, en el fracaso de la sonda espacial MARINER I [2].

La Transformación: Esta mejora podría ser realizada manualmente, lo que resulta inviable en el caso de aplicaciones de miles o cientos de miles de líneas de código fuente. La solución que se propone es utilizar una transformación automática de código fuente la que se ha implementado como una refactorización llamada *"Cambiar a Formato Libre"*. Como se ha explicado, el enfoque utilizado puede parsear correctamente las instrucciones del lenguaje y realizar las validaciones previas y posteriores para asegurar que el comportamiento sea preservado. Dicha transformación ha sido implementada durante este trabajo de investigación. En la Figura 7.9 puede apreciarse el menú del entorno de desarrollo en el cual se encuentra la transformación implementada como una refactorización. Una vista (View) con las diferencias se despliega para previzualizar los cambios del antes y del después, esto puede ser visto en la Figura 7.10. En la Figura 7.11 se puede apreciar el código fuente del programa transformado, en el cual se ha cambiado también los caracteres de comentarios que son los que corresponden al formato libre a partir de Fortran 90 [85, 116]. Una vez realizada dicha transformación se prosigue con la verificación y validación de los resultados. En este caso y debido

a la sencillez del código fuente no ha sido necesario escribir algún tipo de prueba de unidad (unit test), en el caso de haberlas se deberían correr para verificar que el cambio no haya tenido algún impacto no deseado en el software. En la Figura 7.12 pueden verse los resultados de la ejecución del programa tras haber sido sometido a la transformación/cambio. Una vez obtenidos los resultados de la ejecución del programa transformado éstos se comparan con la versión anterior, ver Figura 7.13. Como puede apreciarse, los resultados numéricos no han sido alterados tras la transformación. La primera iteración del proceso ha sido realizada satisfactoriamente. La nueva versión del programa puede verse en la Figura 7.3

```
PROGRAM FIRST
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
DATA N/25/, H/1.0/, X/0.5/
F = SIN(X)
G = COS(X)
    DO 2 I = 1,N
    H = 0.25*H
    D = SIN(X + H) - F
    Q = D/H
    E = ABS(G - Q)
    PRINT *,H,D,Q,E
2   CONTINUE
    STOP
EN
```

Figura 7.3: Código Fuente del Programa FIRST.f90

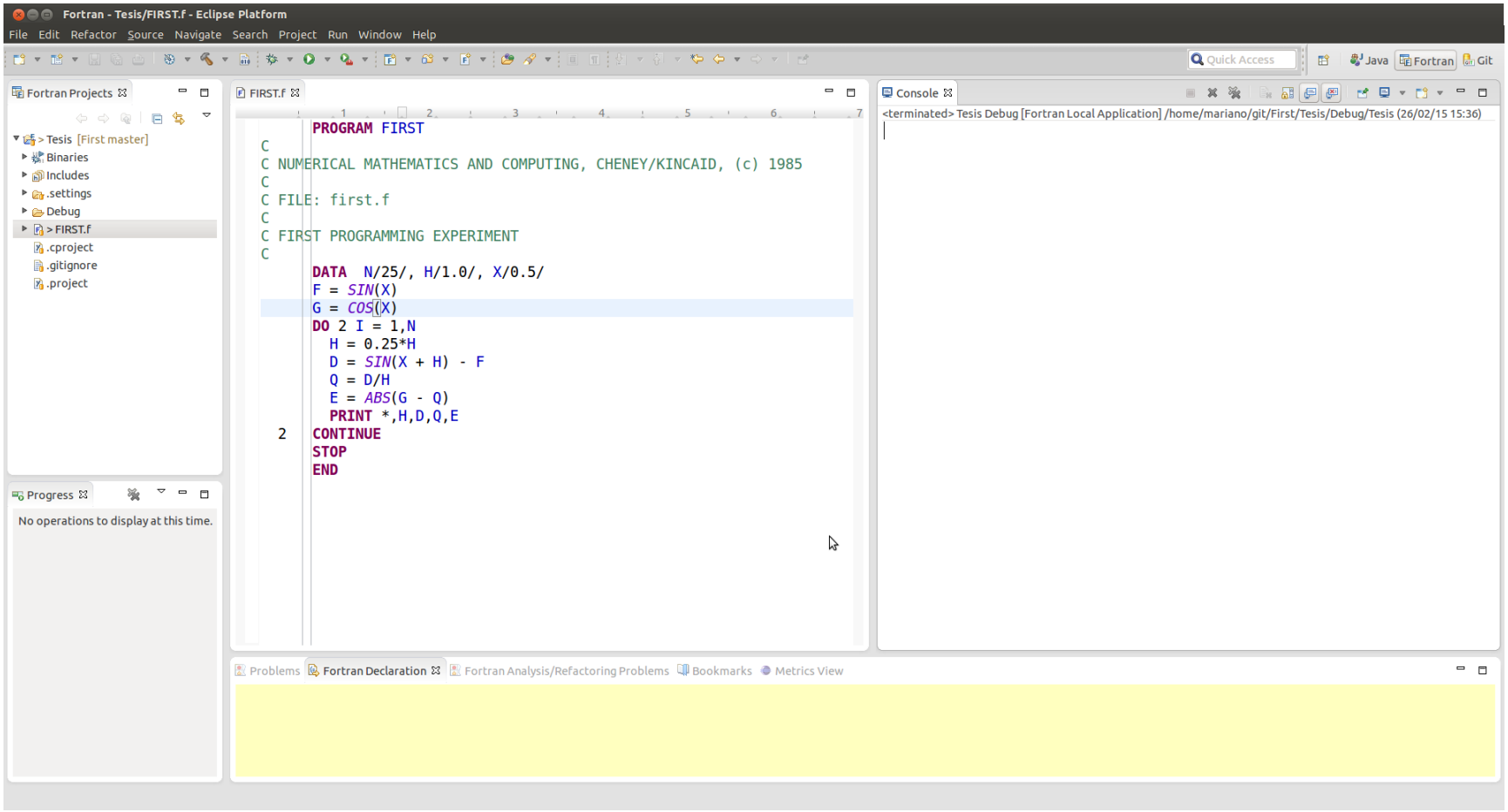


Figura 7.4: Inicio del Proceso de Desarrollo Dirigido por el Cambio para FIRST.F

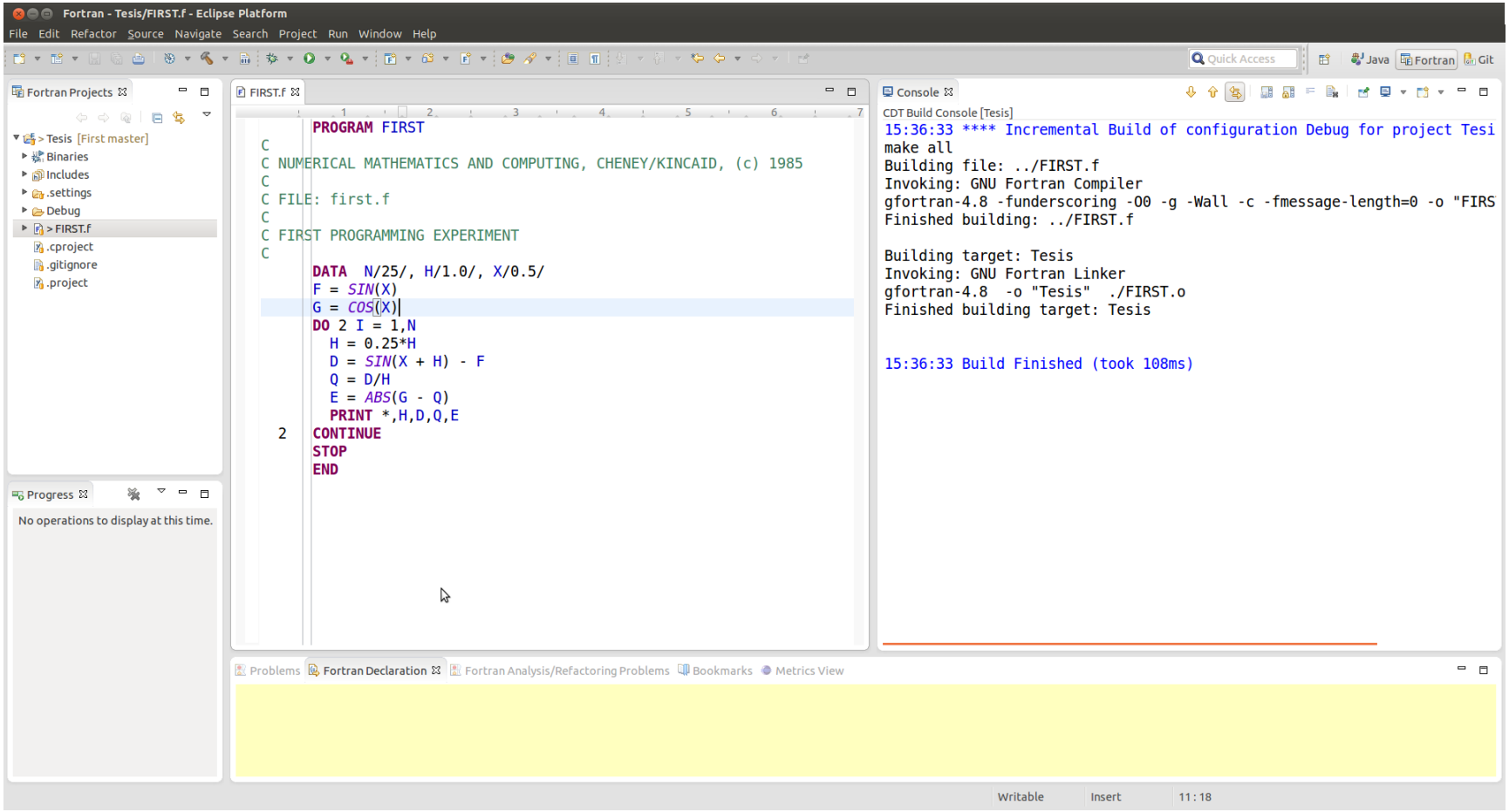


Figura 7.5: Etapa de Built para FIRST.F

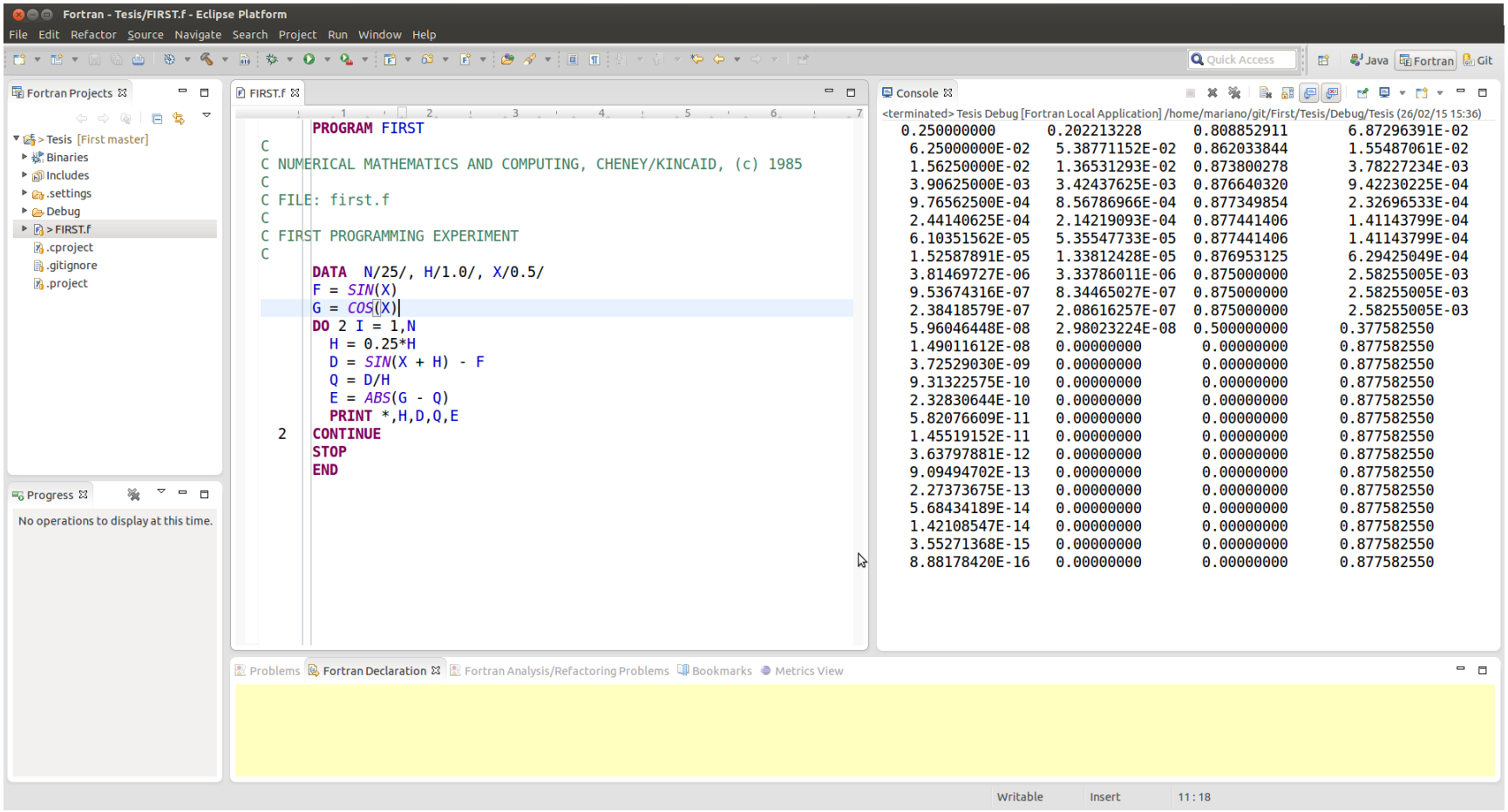


Figura 7.6: Resultados Obtenidos para FIRST.F

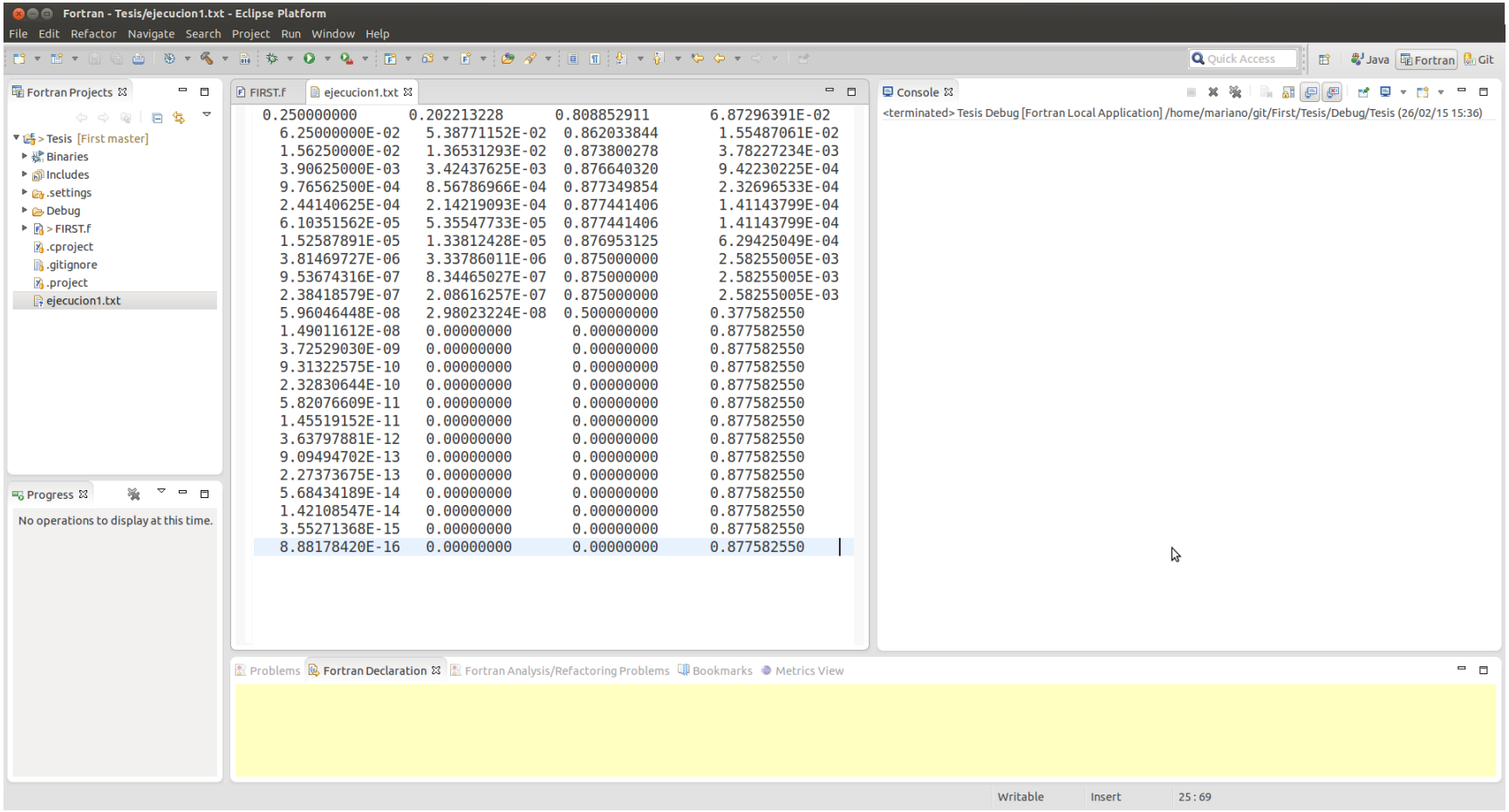


Figura 7.7: Proyecto con los Resultados de la Ejecución

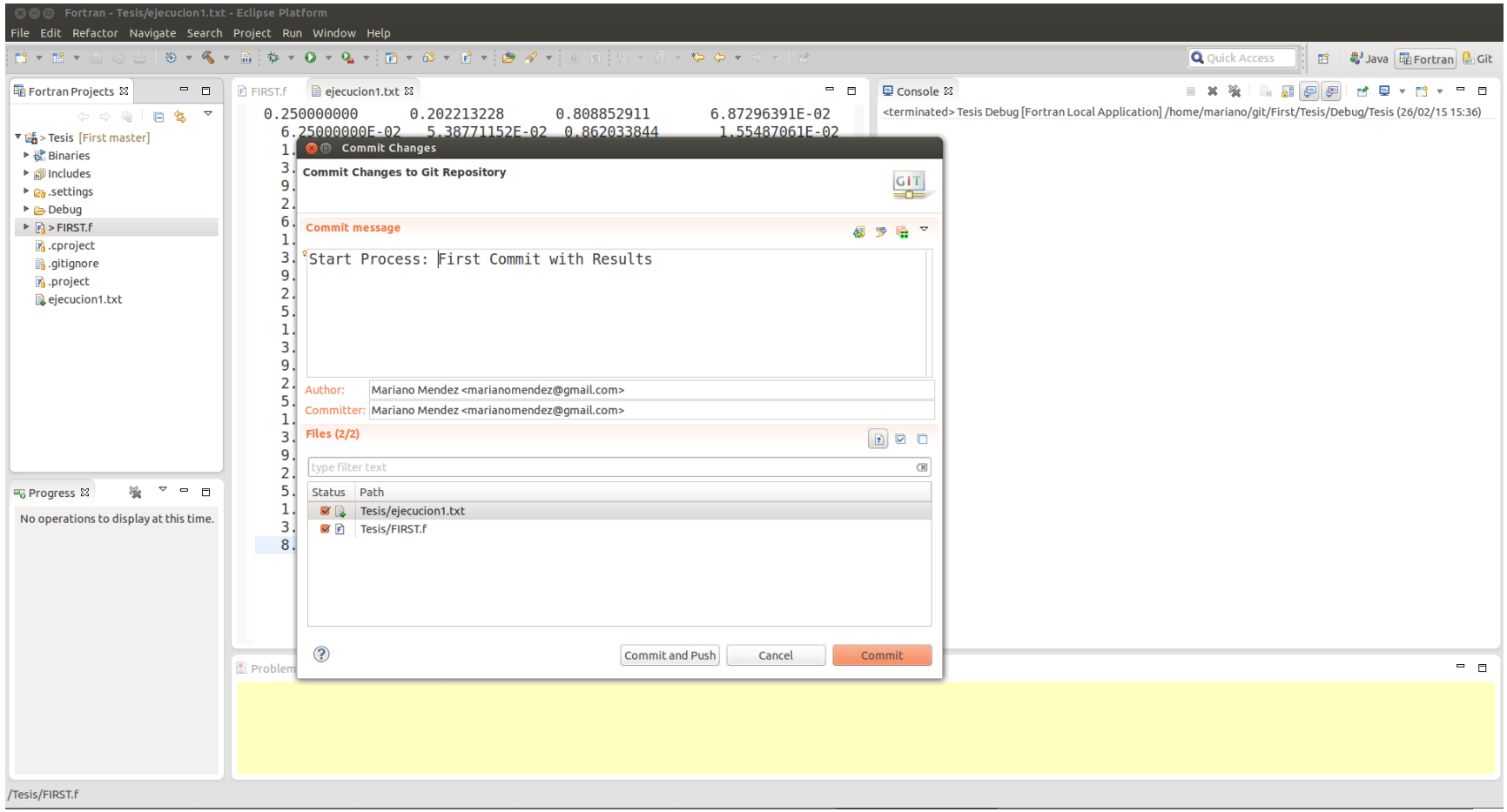


Figura 7.8: Resultados Obtenidos para FIRST.F

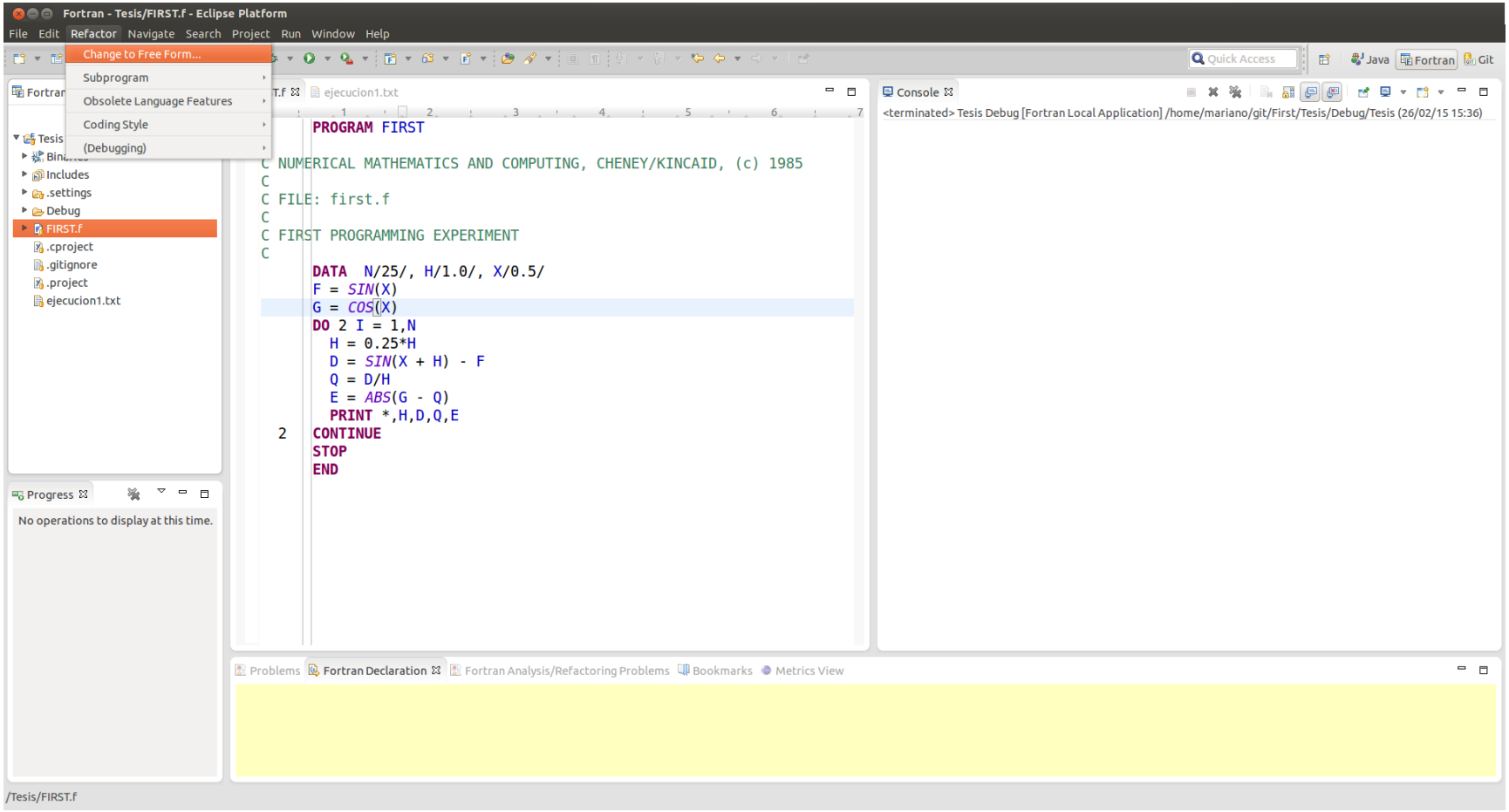


Figura 7.9: Menú en el Cual se Muestra Implementada la Transformación para Pasar a Formato Libre

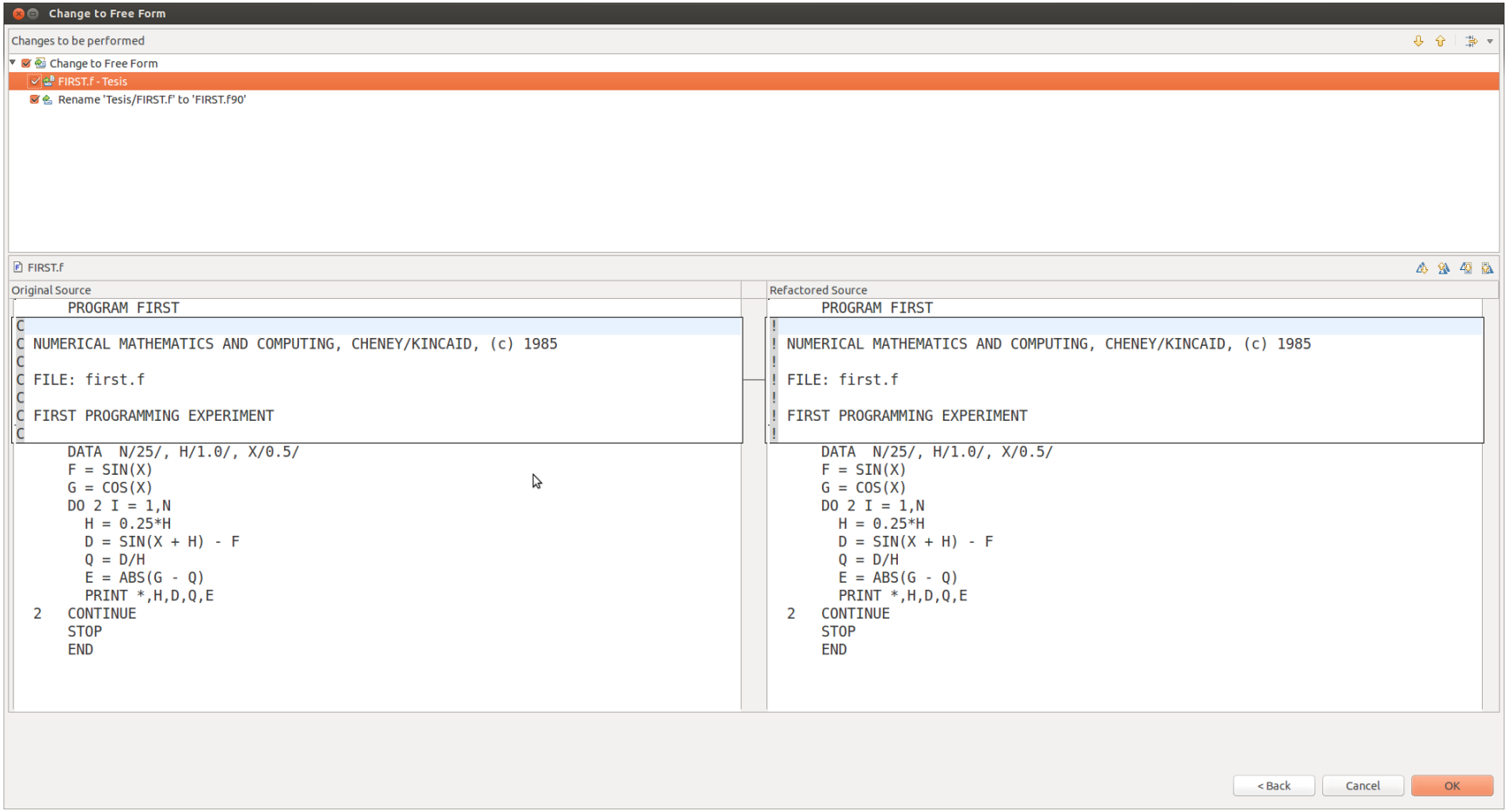


Figura 7.10: Vista de Diferencia (DIFF VIEW) en la que se Aprecia el Antes y el Después de la Aplicación de la Transformación de Código Fuente

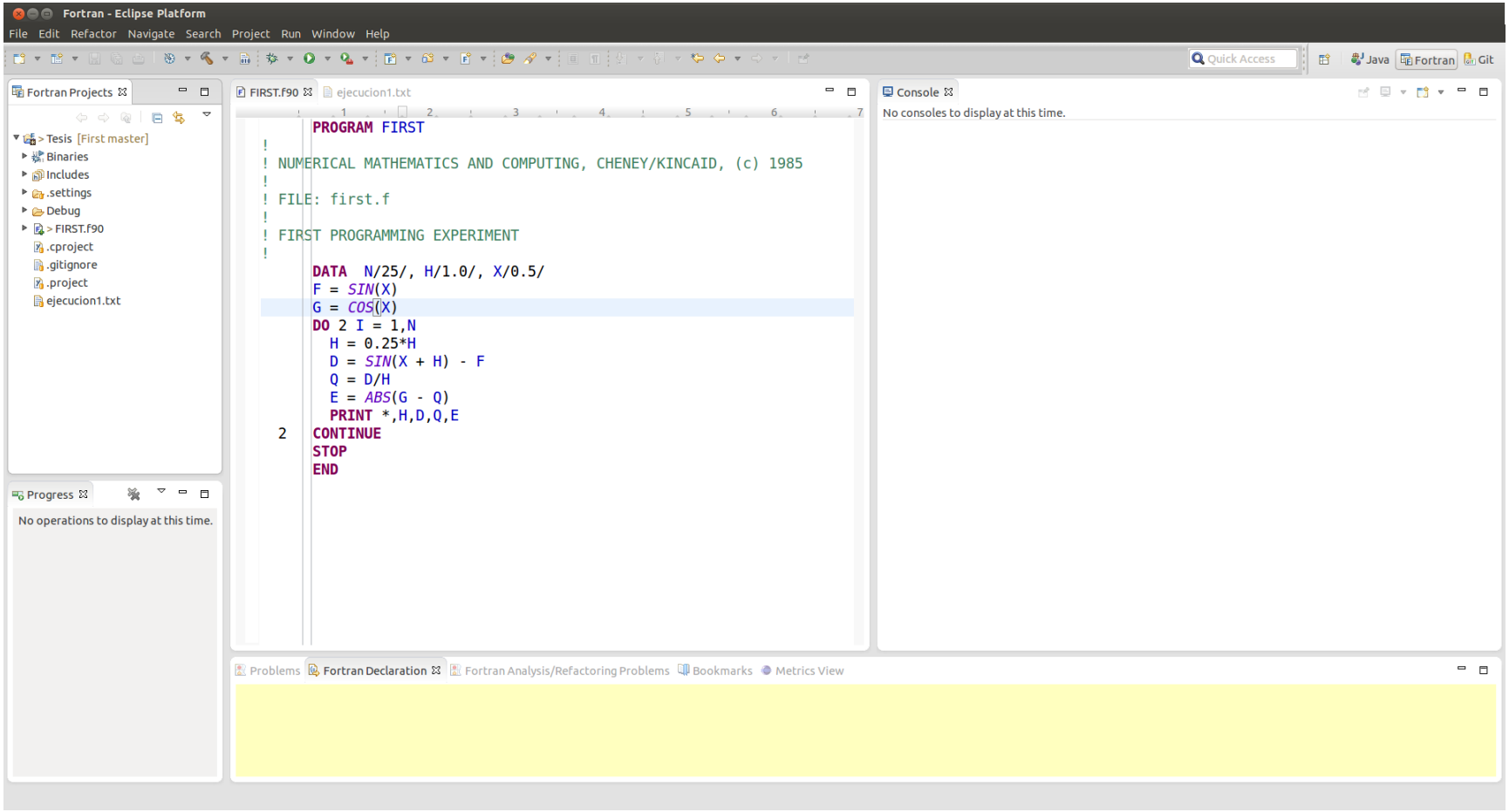


Figura 7.11: El Programa FIRST.F ha Sido Transformado en FIRST.f90

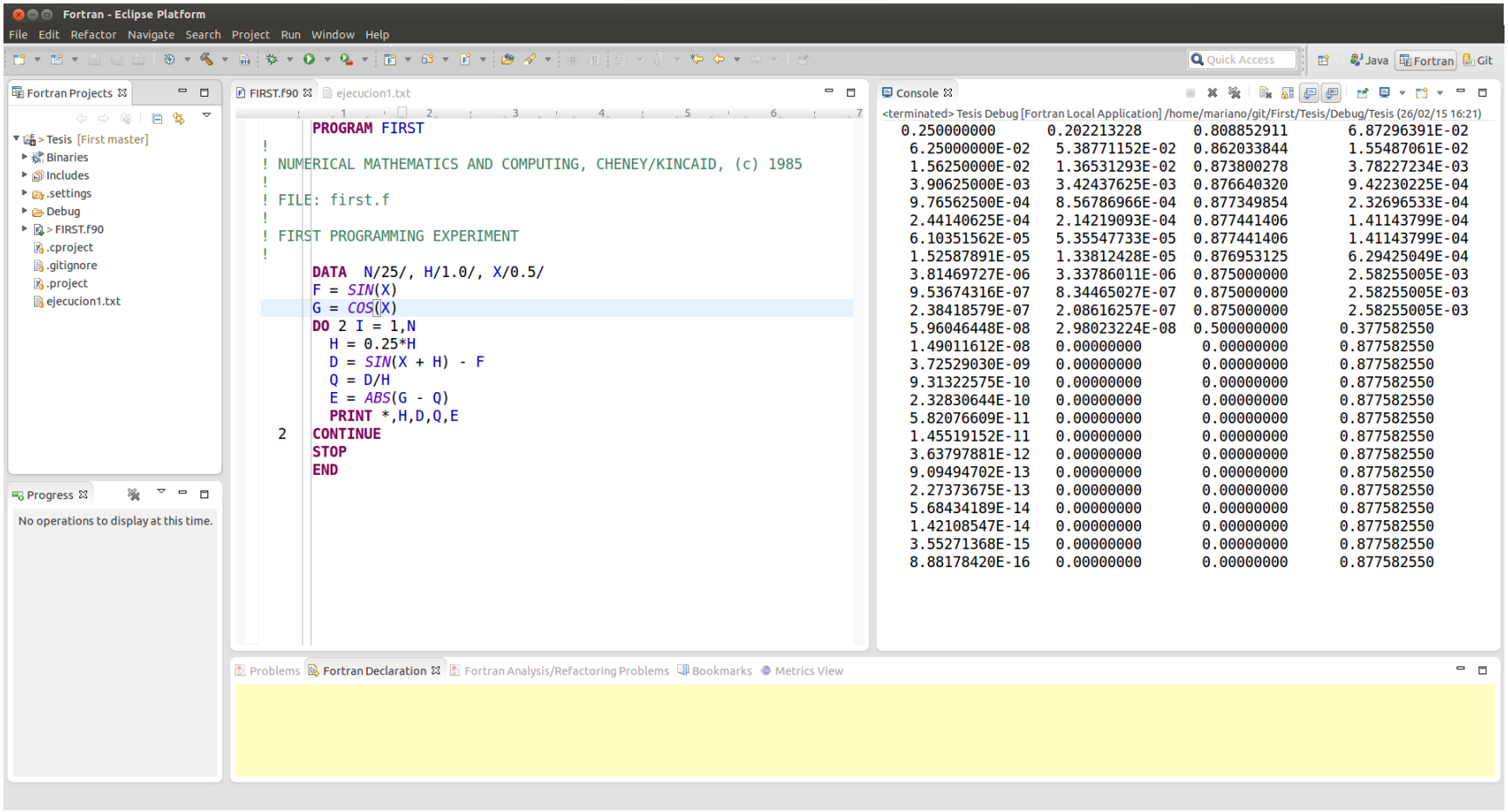


Figura 7.12: Resultados Obtenidos en la Corrida de FIRST.f90

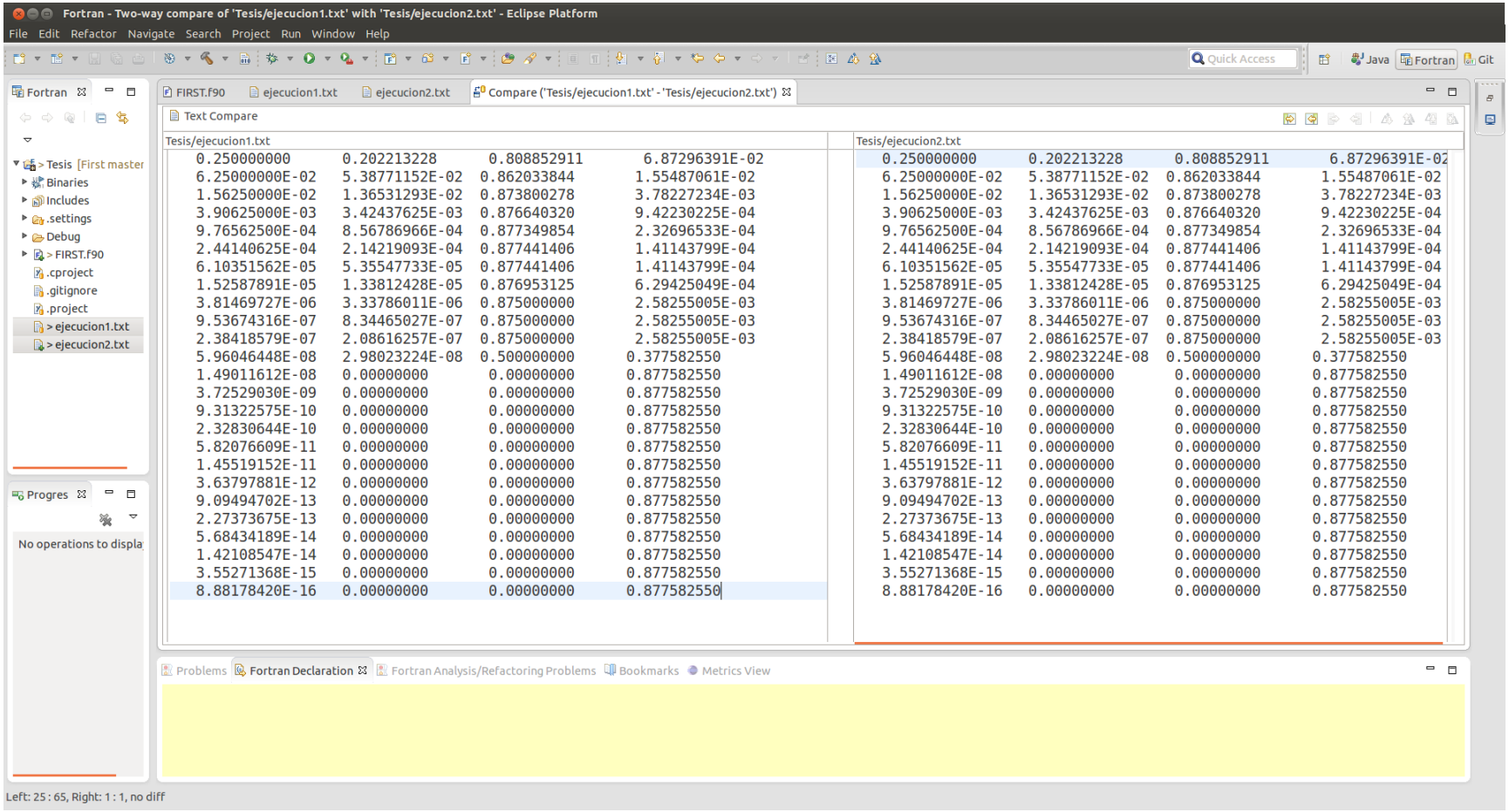
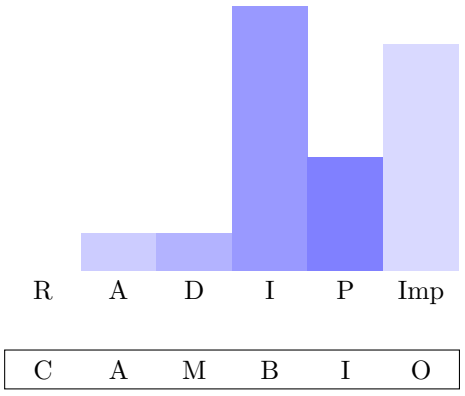


Figura 7.13: Se Comparan los Resultados de las Dos Versiones del Programa, la Original FIRST.f vs. la Transformada FIRST.f90

Iteración 2: Cambiar a minúsculas las instrucciones del lenguaje

De la misma forma que en el apartado anterior establecemos una versión inicial que justamente es la salida del ciclo anterior, y a partir de ésta iniciamos el proceso de desarrollo dirigido por el cambio número 2 de la lista. En este caso debido a la simplicidad del software la fase de comprensión es extremadamente sencilla,al tratarse de un solo archivo de 20 líneas de código fuente, pues no es necesario obtener un árbol de llamadas estático, ni el cálculo de métricas para determinar la prioridad de aplicación del cambio, etc. Por otro lado el conjunto de actividades centrales del desarrollo de software también está distribuido de forma sencilla ver Figura 7.14



(b) Actividades Centrales

Figura 7.14: Carga/Distribución de las Actividades Centrales del Desarrollo de Software para el Cambio 2

Otra vez se repite el flujo de trabajo propuesto en el Capítulo 3:

1. Establecer una versión inicial del código fuente
2. Transformar el código fuente
3. Verificar el código fuente obtenido
4. Validar los resultados numéricos

5. Aceptar/Rechazar el cambio en base a los resultados numéricos

6. Documentar

Se establece como la versión inicial la salida de la iteración anterior, en este caso la iteración 1. La transformación a realizar es la de cambiar a minúsculas las instrucciones del lenguaje. Dado que en las versiones más antiguas de Fortran los programadores estaban obligados a utilizar letras mayúsculas en los programas, en la actualidad gran cantidad de programadores siguen utilizando las mayúsculas en sus programas [217]. Esta obligación ha sido eliminada de los estándares de Fortran por lo menos desde Fortran 90 [9]. Teniendo en cuenta que las instrucciones del lenguaje Fortran no son palabras reservadas, y pueden encontrarse variables que estén nombradas como una instrucción, esta transformación no puede ser realizada utilizando técnicas de pattern matching o de búsqueda y reemplazo (Search and Replace). Para ello se utiliza una transformación implementada como refactorización que realiza esta tarea recorriendo el AST del programa y cambiando solamente de mayúsculas a minúsculas las instrucciones ver Figura 7.16, Figura 7.15, Figura 7.17, Figura 7.18 y Figura 7.19.

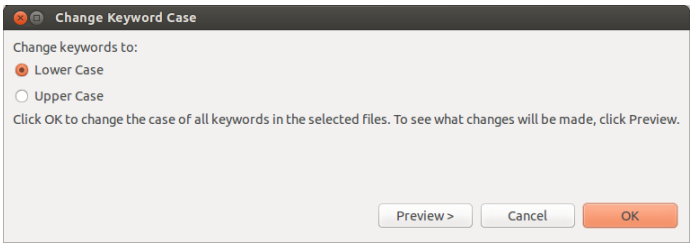


Figura 7.15: UI para Seleccionar el Tipo de Cambio

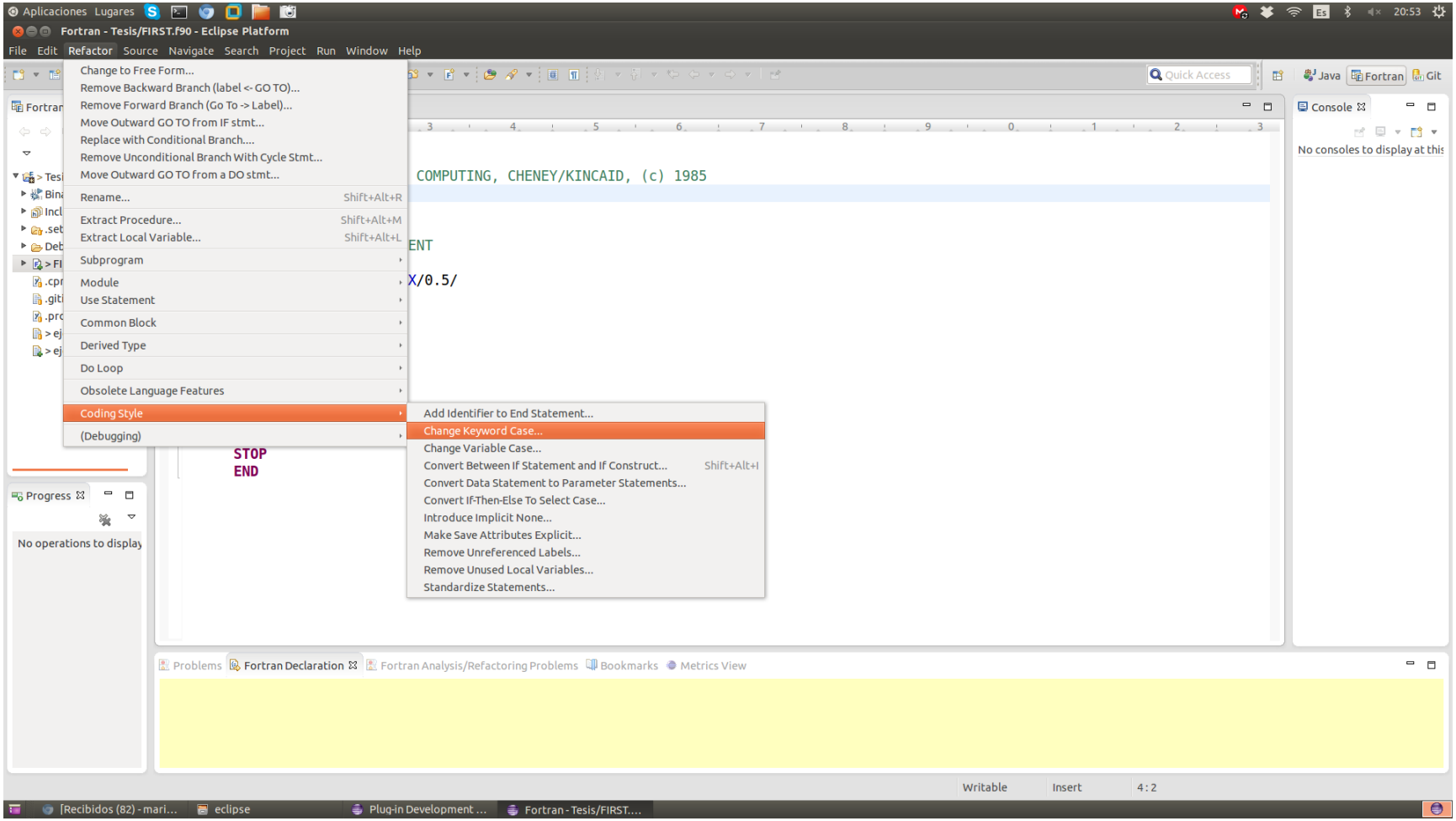


Figura 7.16: Opción de Menú de Transformaciones del IDE

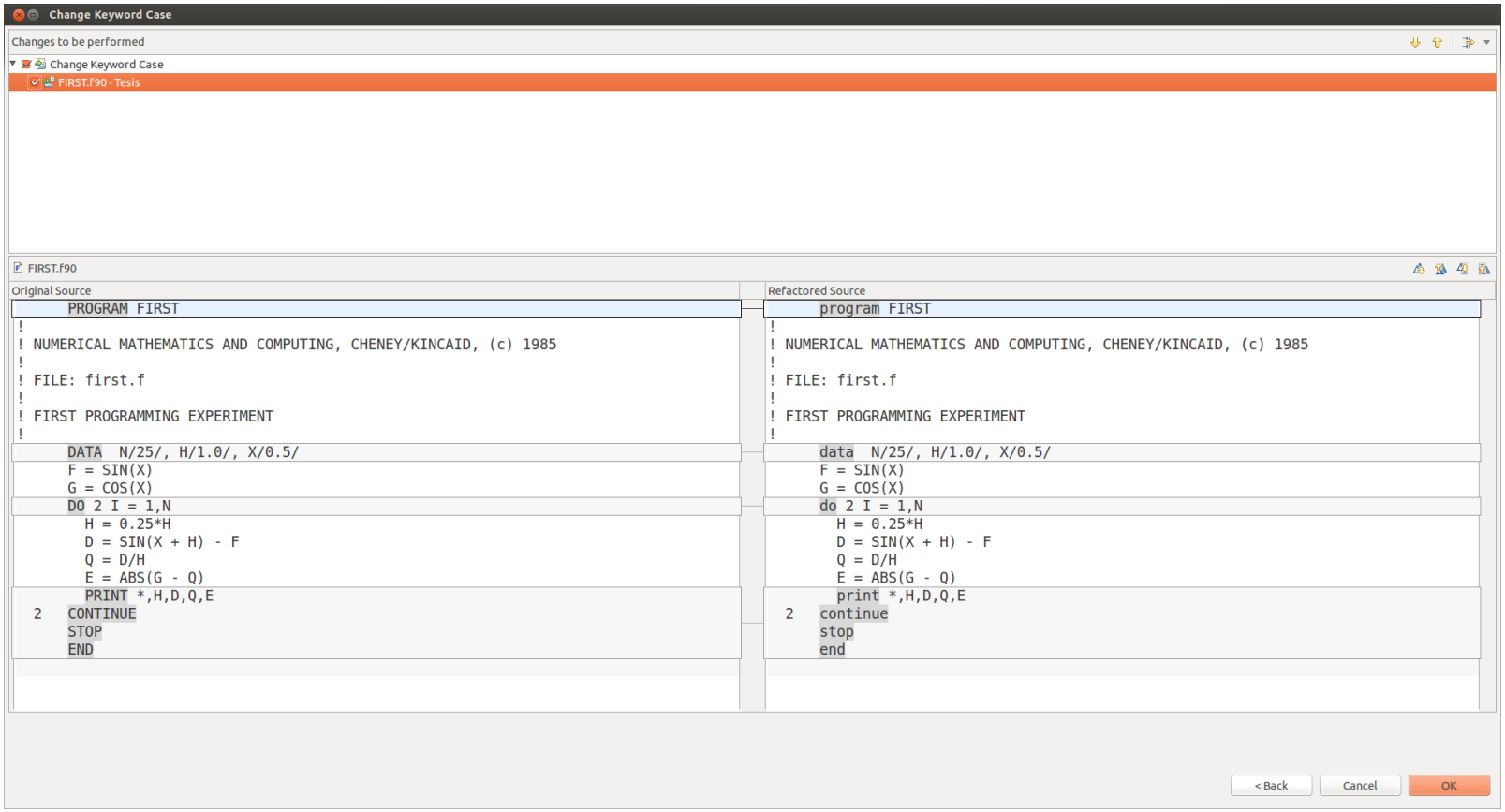


Figura 7.17: Vista de Diferencia (DIFF VIEW) en la que se Aprecia el Antes y el Después de la Aplicación de la Transformación de Código Fuente Iteración 2

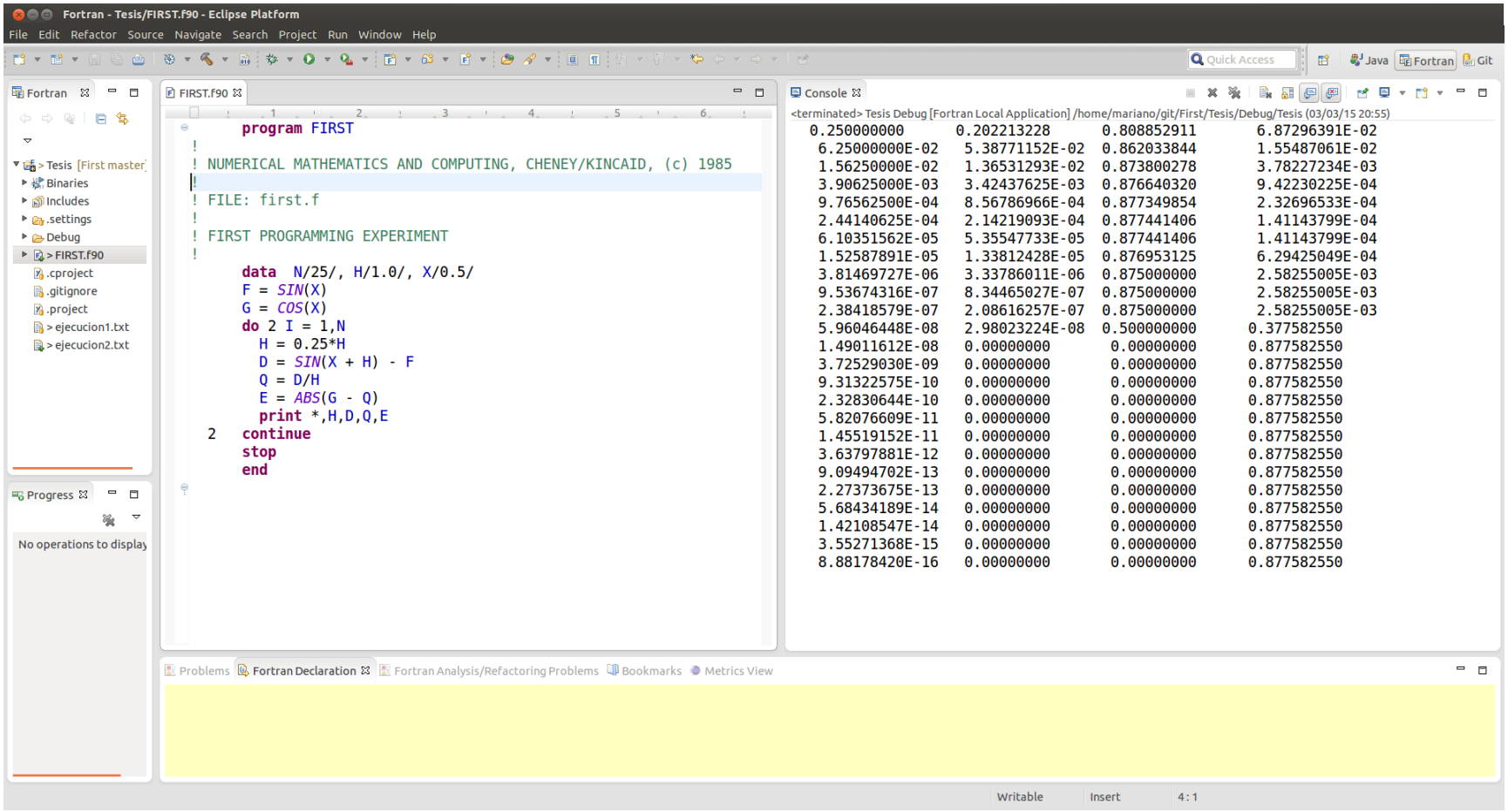


Figura 7.18: Resultados Obtenidos en la Corrida de FIRST.f90 en la Segunda Iteración

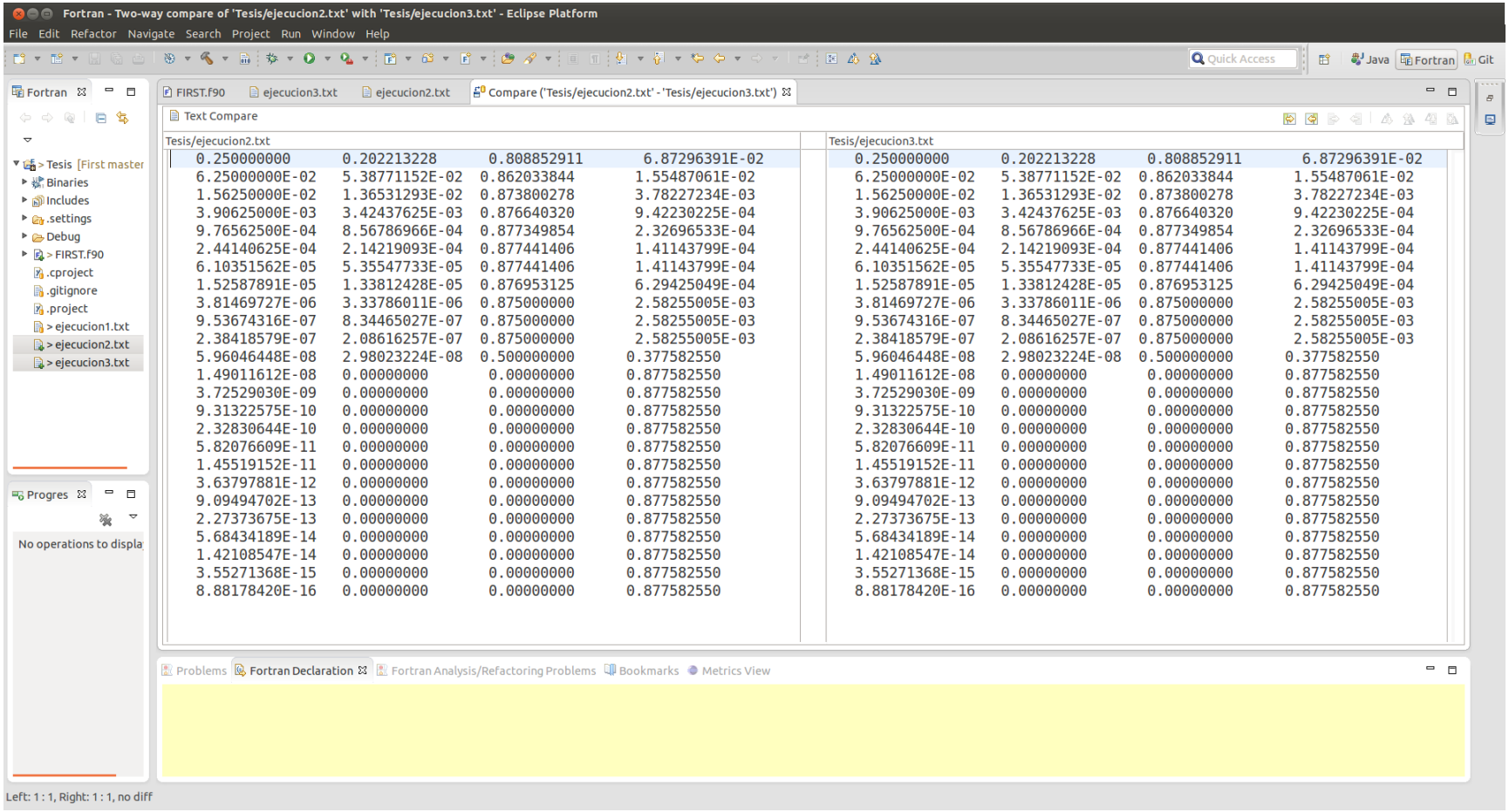


Figura 7.19: Se Comparan los Resultados de las Dos Versiones del Programa, la Original FIRST.f vs. la Transformada FIRST.f90

Iteración 3: Cambiar a minúsculas los nombres de los identificadores

El tercer cambio tiene que ver con pasar a minúsculas los nombres de los identificadores, esto no es obligatorio. Para ello seguimos siempre el flujo de trabajo propuesto y congelamos la versión del código que tenemos como versión de trabajo, en este caso es la salida de la iteración 2. Para ello utilizamos el software de versionado Git y su plugin para Eclipse, ver Figura 7.20, Figura 7.21 y Figura 7.22.

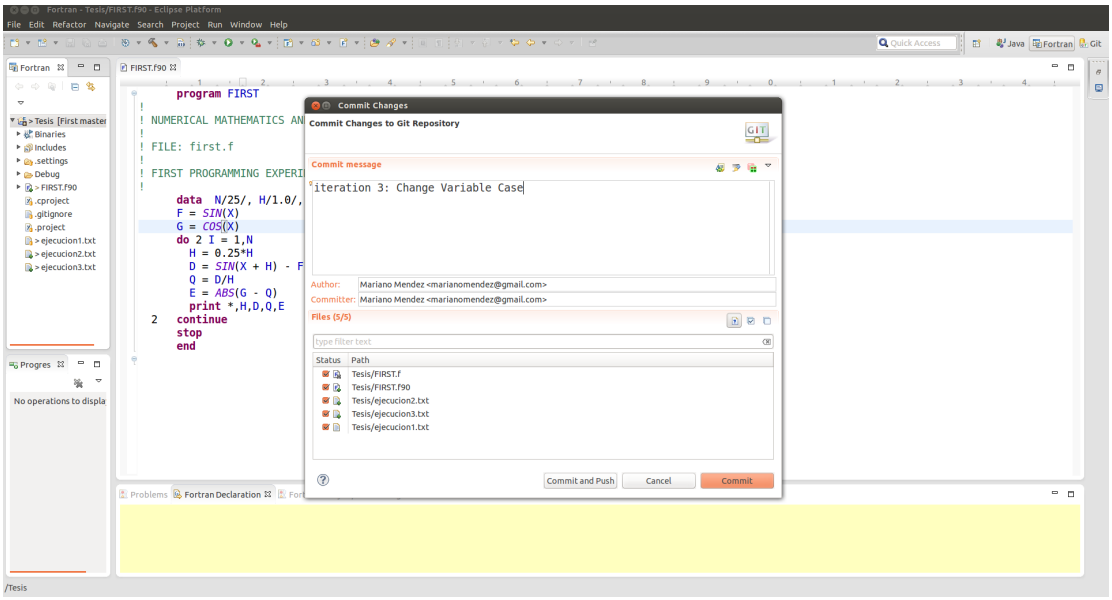


Figura 7.20: Vista de la Ventana de Commit del Plugin de Eclipse de Git

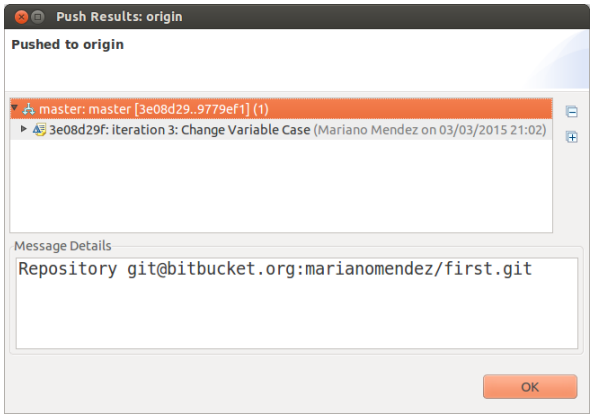


Figura 7.21: Resultado del Push de la Versión Transformada FIRST.f90

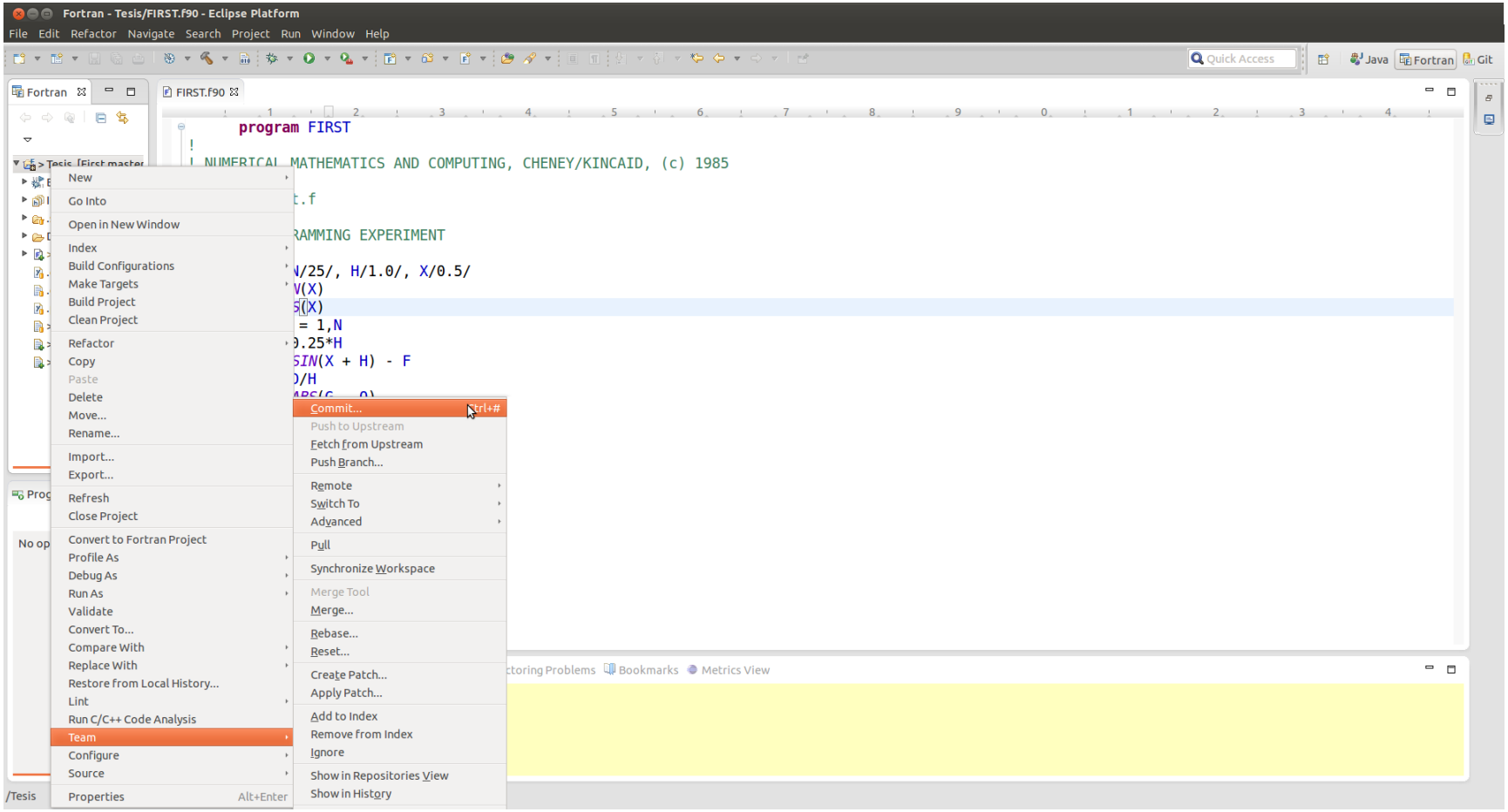


Figura 7.22: Commit de la Versión de Trabajo

Una vez realizada esta operación se prosigue a la aplicación del cambio. En este caso al igual que en la iteración anterior se debe usar el abordaje del AST para poder determinar cuáles son los identificadores y cuáles las instrucciones o expresiones literales. Esta transformación ha sido implementada en el ámbito de este trabajo de investigación y es un aporte a la distribución oficial de Photran 8.1. Una vez establecido el cambio y realizado el mismo, ver Figura 7.25 y Figura 7.28 se procede a realizar la compilación y ejecución del programa para validar y verificar los resultados numéricos obtenidos, de existir un conjunto de pruebas deberían también correrse, ver Figura 7.28 y Figura 7.29. Se ha de observar que no existe variación alguna en los resultados numéricos obtenidos después de haber aplicado 3 cambios. Se ha detectado un pequeño error en dicha transformación que consiste en que las variables que aparecen en el bloque de la instrucción DATA no son transformadas. El código resultante hasta el momento se ve en la Figura 7.23.

```
program FIRST
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
     data N/25/, H/1.0/, X/0.5/ <--bug en la transformaci n
     f = SIN(x)
     g = COS(x)
     do 2 i = 1,n
        h = 0.25*h
        d = SIN(x + h) - f
        q = d/h
        e = ABS(g - q)
        print *,h,d,q,e
2    continue
     stop
end
```

Figura 7.23: Código Fuente del Programa FIRST.f90 Tras Haber Sido Realizada la Iteración 3

Debido al error reportado hemos realizado a mano el cambio a minúsculas del bloque DATA ver Figura 7.24.

```
program FIRST
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
      data  n/25/, h/1.0/, x/0.5/
      f = SIN(x)
      g = COS(x)
      do 2 i = 1,n
           h = 0.25*h
           d = SIN(x + h) - f
           q = d/h
           e = ABS(g - q)
           print *,h,d,q,e
2     continue
stop
end
```

Figura 7.24: Código Fuente del Programa FIRST.f90

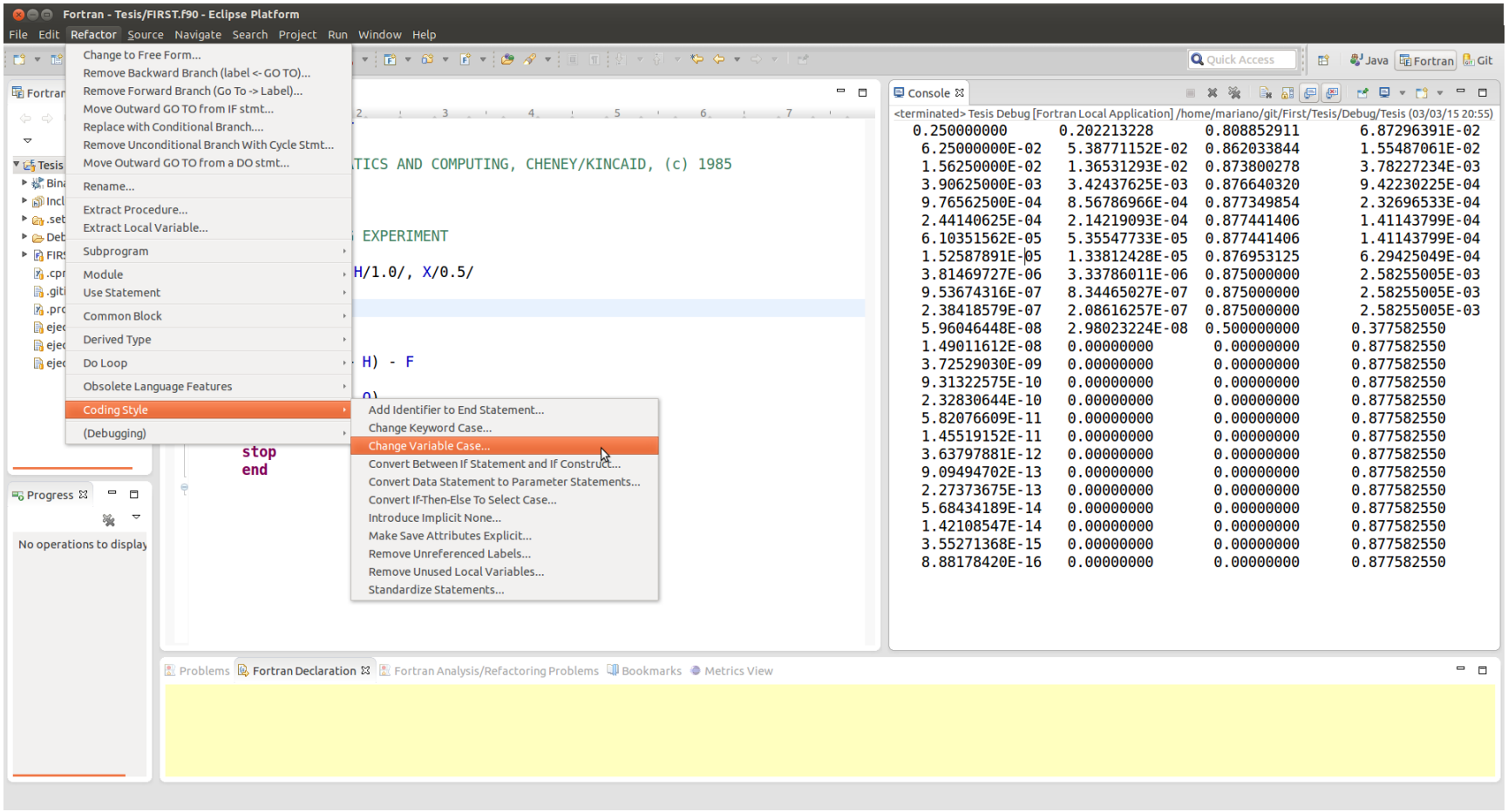


Figura 7.25: Se Comparan los Resultados de las Dos Versiones del Programa, la Priginal FIRST.f vs la Transformada FIRST.f90

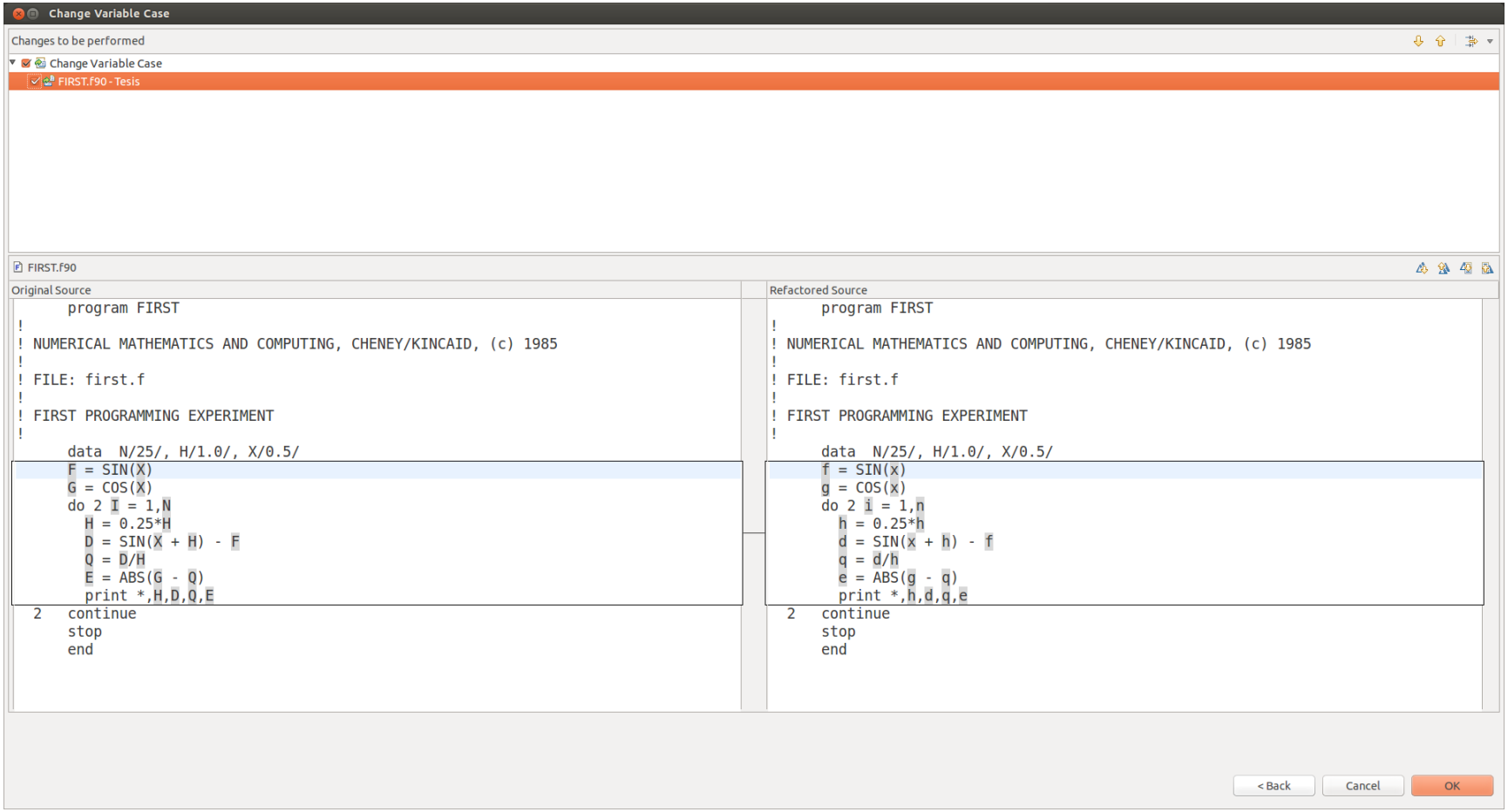


Figura 7.26: Vista de Diferencia (DIFF VIEW) en la que se Aprecia el Antes y el Después de la Aplicación de la Transformación de Código Fuente Iteración 3

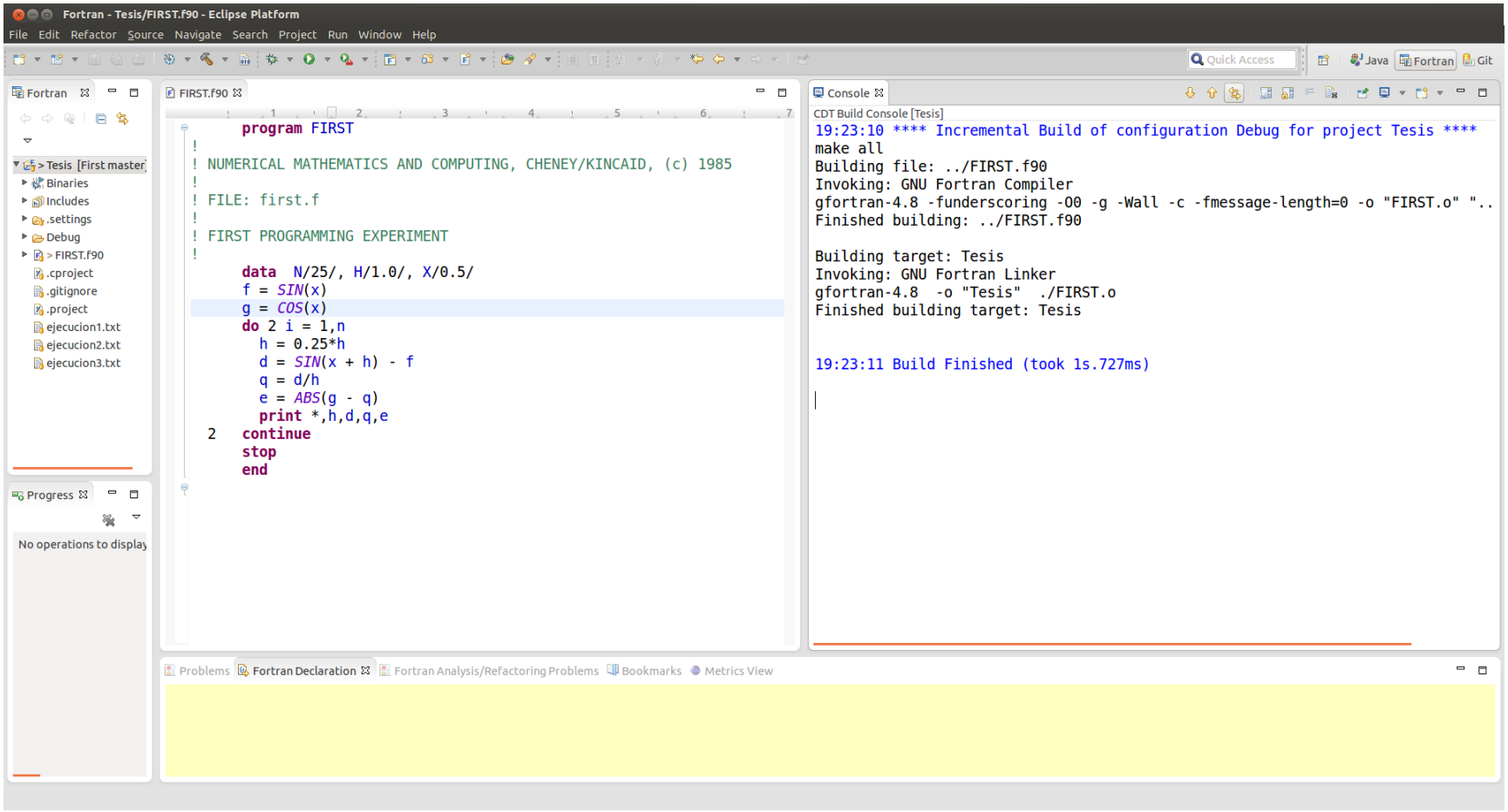


Figura 7.27: Resultado de la Compilación Después de la Aplicación de la Transformación de Código Fuente Iteración 3

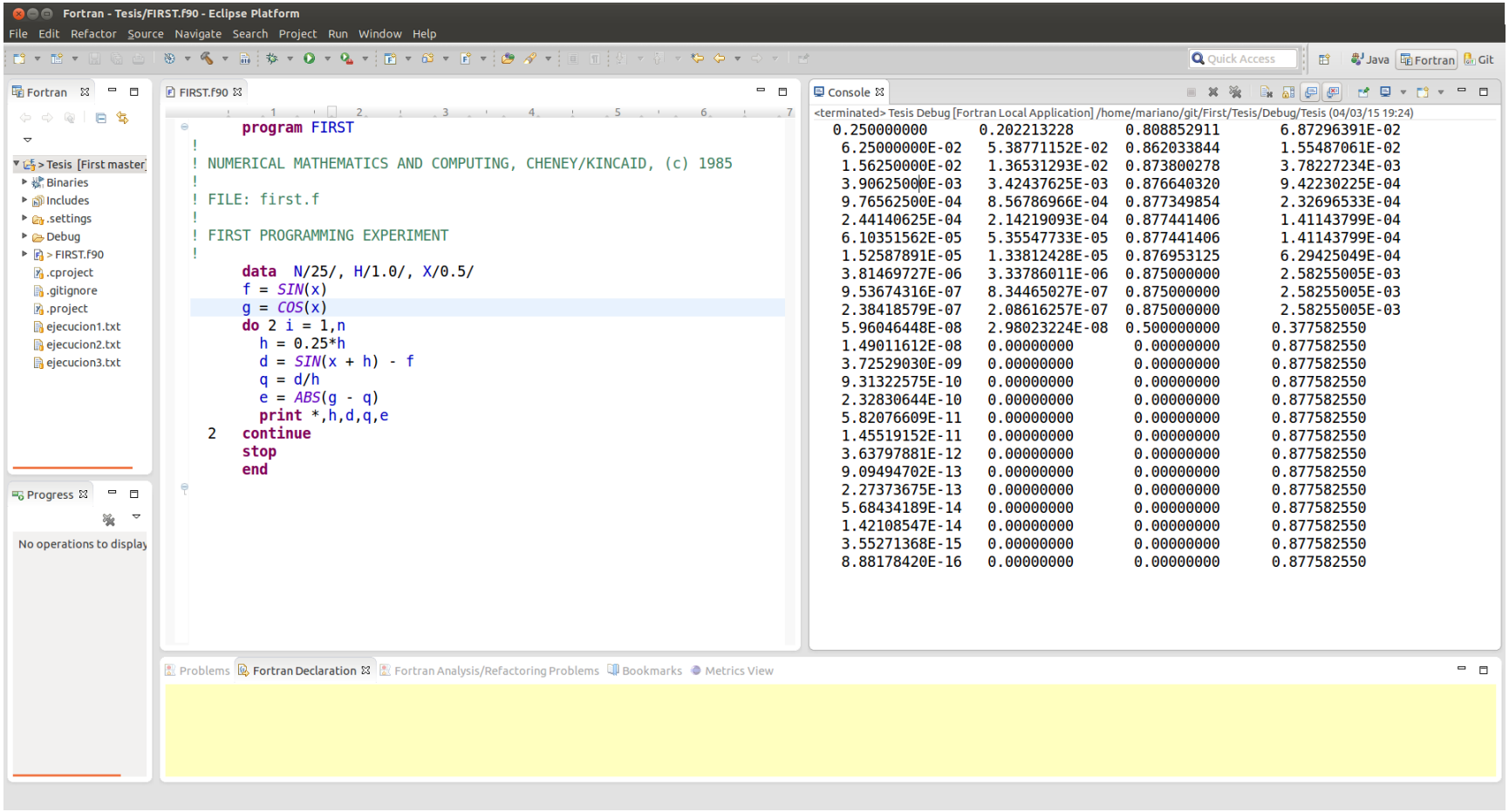


Figura 7.28: Vista de los Resultados Obtenidos Después de la Aplicación de la Transformación de Código Fuente Iteración 3

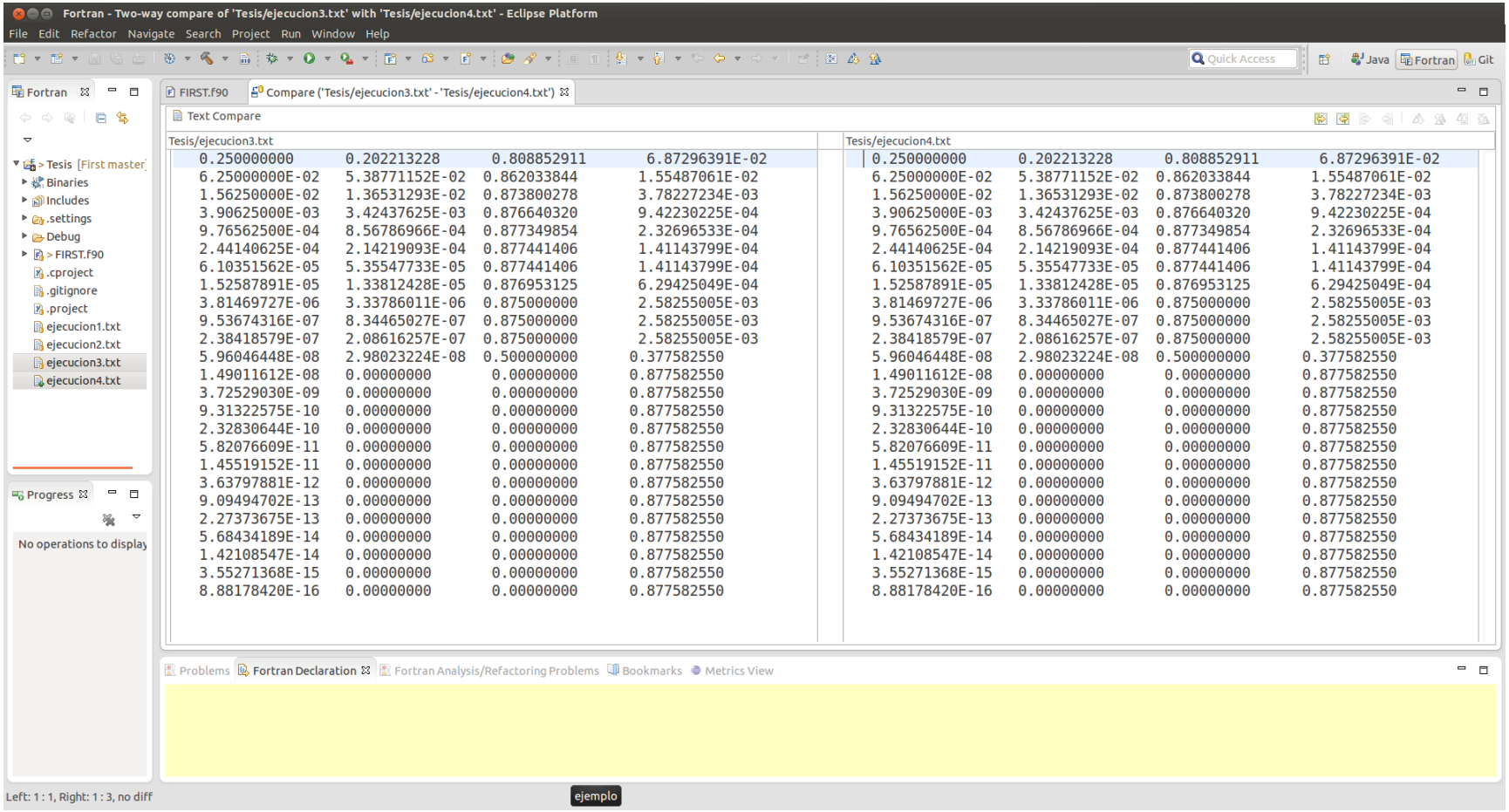


Figura 7.29: Vista de Diferencia (DIFF VIEW) de la Comparación de los Resultados de la Ejecución del Código Resultante de la Iteración 2 con el de la Iteración 3

Iteración 4: Introducir la instrucción IMPLICIT NONE

Ya en el estándar de FORTRAN 66 está permitido la declaración implícita de variables, es decir que el nombre de la variable sirve como identificador y a su vez define el tipo de dato de la misma, “The name employed to identify a datum or function carries the data type assosiation” Sección 4.1 del estándar de FORTRAN 66 [87]. Existen varios inconvenientes que acarrea la declaración implícita de variables, por ejemplo el agregado de errores debido a inconsistencias de tipeo, de asignación de tipos de datos, control de rangos y valores, entre otros. Es recomendable que todas las variables sean declaradas implícitamente, para ello existe la instrucción Fortran IMPLICIT NONE, que obliga a declarar todas las variables explícitamente, es decir, especificando su tipo de dato. Realizar esta tarea manualmente es un proceso largo, tedioso y propenso a cometer errores, para ello se aplicará una transformación implementada como una refactorización para Fortran, la cual se ocupa automáticamente de realizar este cambio/transformación mediante el VPG (Virtual Program Graph) en el cual se encuentra la tabla de símbolos, en ésta se encuentra la información necesaria para determinar si un token es un identificador, de qué tipo es y si está declarado implícitamente o no y finalmente la posterior modificación del AST. Para ello se utiliza como en los casos anteriores el flujo de trabajo propuesto, ver Figura 7.33. Luego se procede a realizar la transformación cambio, ver Figura 7.34 y Figura 7.35. Se procede a la compilación y ejecución del código fuente resultante, ver Figura 7.36 y Figura 7.37. Y por último se verifican los resultados obtenidos mediante la comparación de los resultados numéricos, ver Figura 7.38. Código fuente resultante de la transformación aplicada ver la Figura 7.30. Tras haber realizado la transformación se ha notado que las variables han sido declaradas una por línea, tal vez sería interesante proponer una transformación que permita agrupar las declaraciones en una única línea, ver Figura 7.31. Esta transformación podría llamarse "Group Variable Declaration". Por ello en esta iteración también se cambiará, manualmente lo detectado anteriormente 7.32.

```
program FIRST
      implicit none
      real :: d
      real :: e
      real :: f
      real :: g
      real :: h
      integer :: i
      integer :: n
      real :: q
      real :: x
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
      data n/25/, h/1.0/, x/0.5/
      f = SIN(x)
      g = COS(x)
      do 2 i = 1,n
            h = 0.25*h
            d = SIN(x + h) - f
            q = d/h
            e = ABS(g - q)
            print *,h,d,q,e
2     continue
stop
end
```

Figura 7.30: Código Fuente del Programa FIRST.f90 Resultado de la Transformación

```
      implicit none                 implicit none
      real :: d                     real:: d,e,f,g,h,q,x
      real :: e                     integer :: i,n
      real :: f
      real :: g          =>
      real :: h
      integer :: i
      integer :: n
      real :: q
      real :: x
```

Figura 7.31: Código Fuente del Programa FIRST.f90 con la Propuesta de Agrupación

```
program FIRST
      implicit non
      real:: d,e,f,g,h,q,x
      integer :: i,n
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
      data  n/25/, h/1.0/, x/0.5/
      f = SIN(x)
      g = COS(x)
      do 2 i = 1,n
         h = 0.25*h
         d = SIN(x + h) - f
         q = d/h
         e = ABS(g - q)
         print *,h,d,q,e
2     continue
      stop
end
```

Figura 7.32: Código Fuente del Programa FIRST.f90 Resultado Final de la Transformación de la Iteración 4

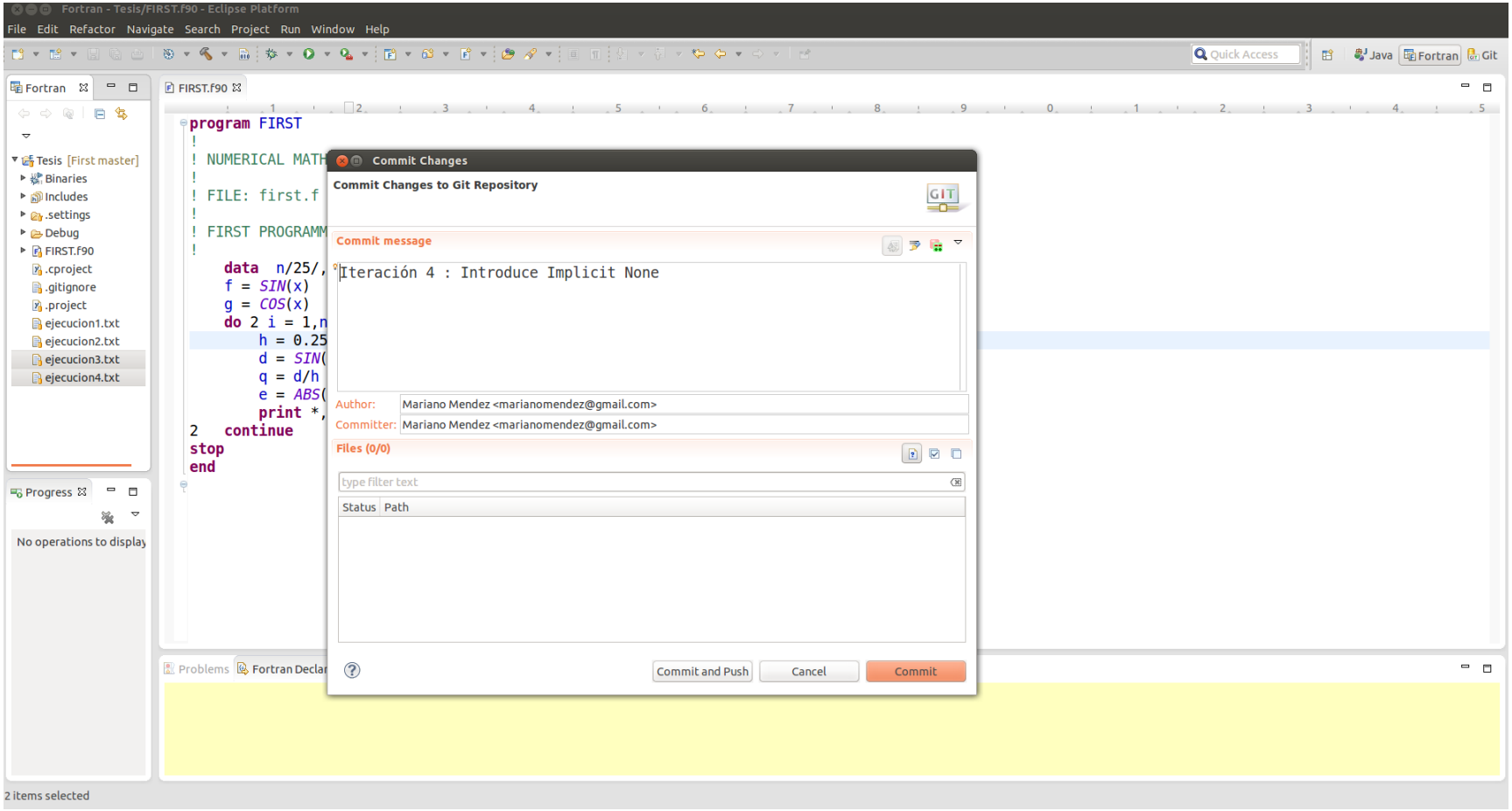


Figura 7.33: Commit de la Versión de Trabajo

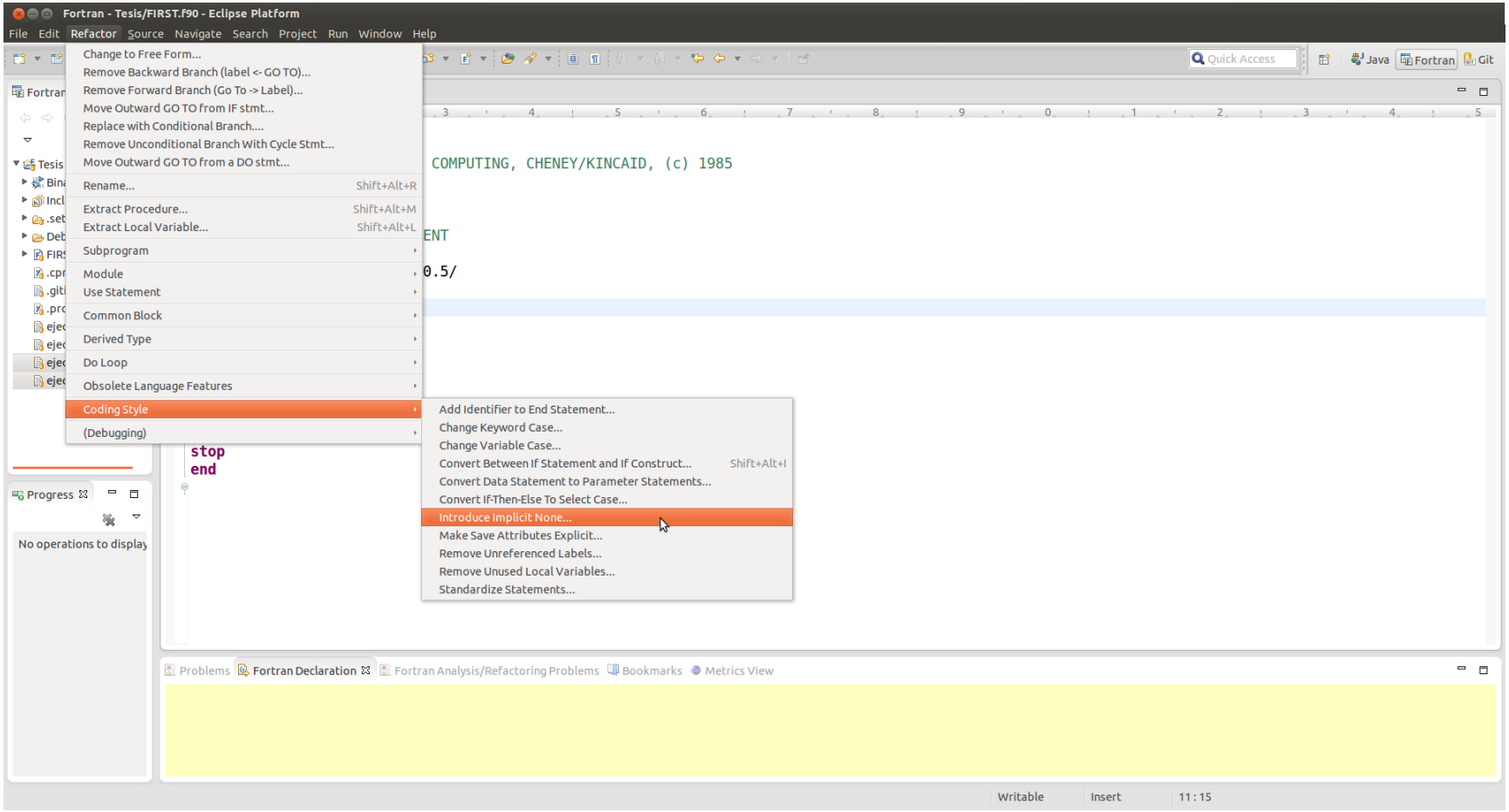


Figura 7.34: Selección de la Transformación en el Menú de la Aplicación

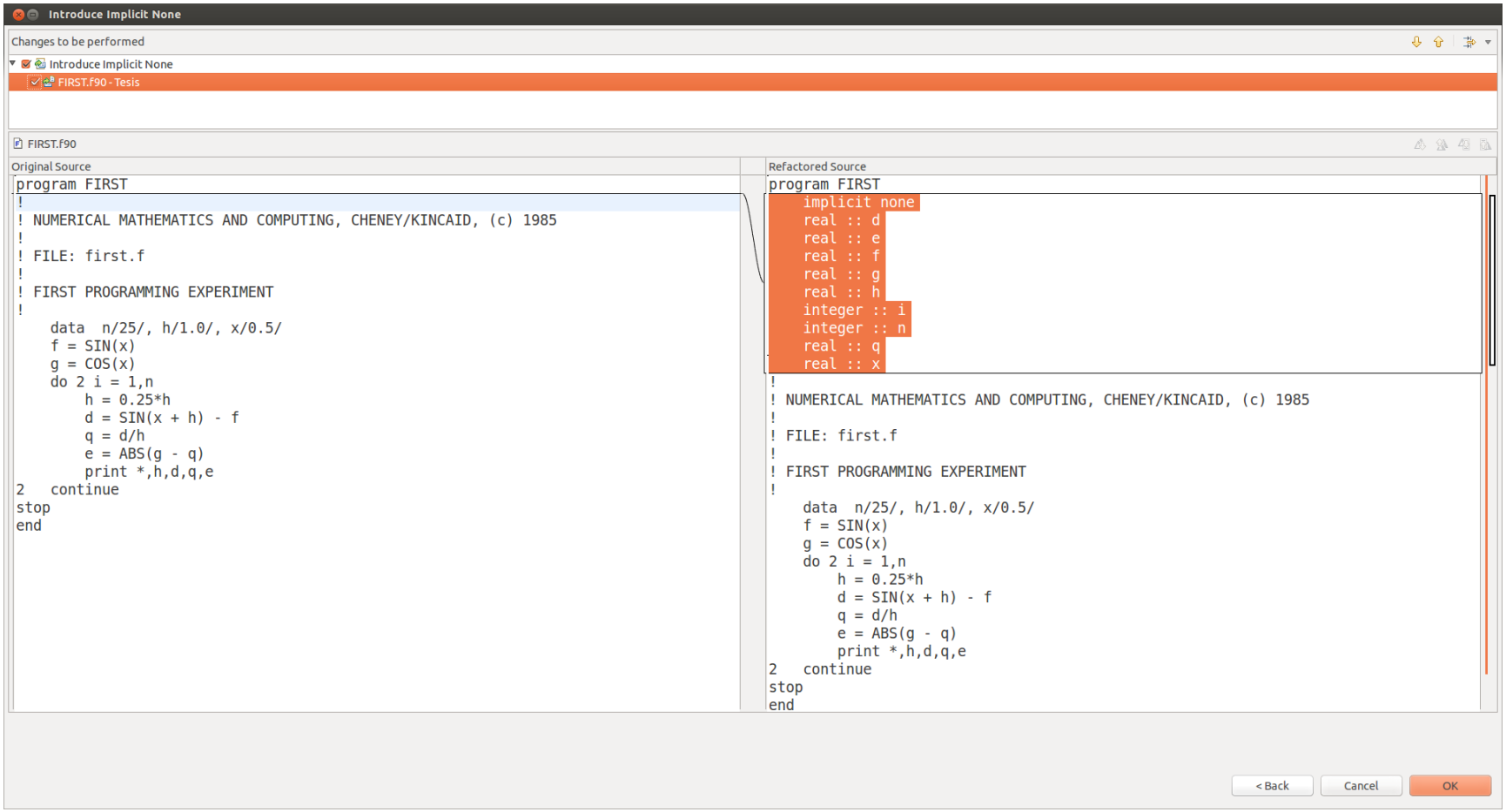


Figura 7.35: Vista Previa en Diferencia (DIFF VIEW) del Código Resultante de la Transformación de la Iteración 4

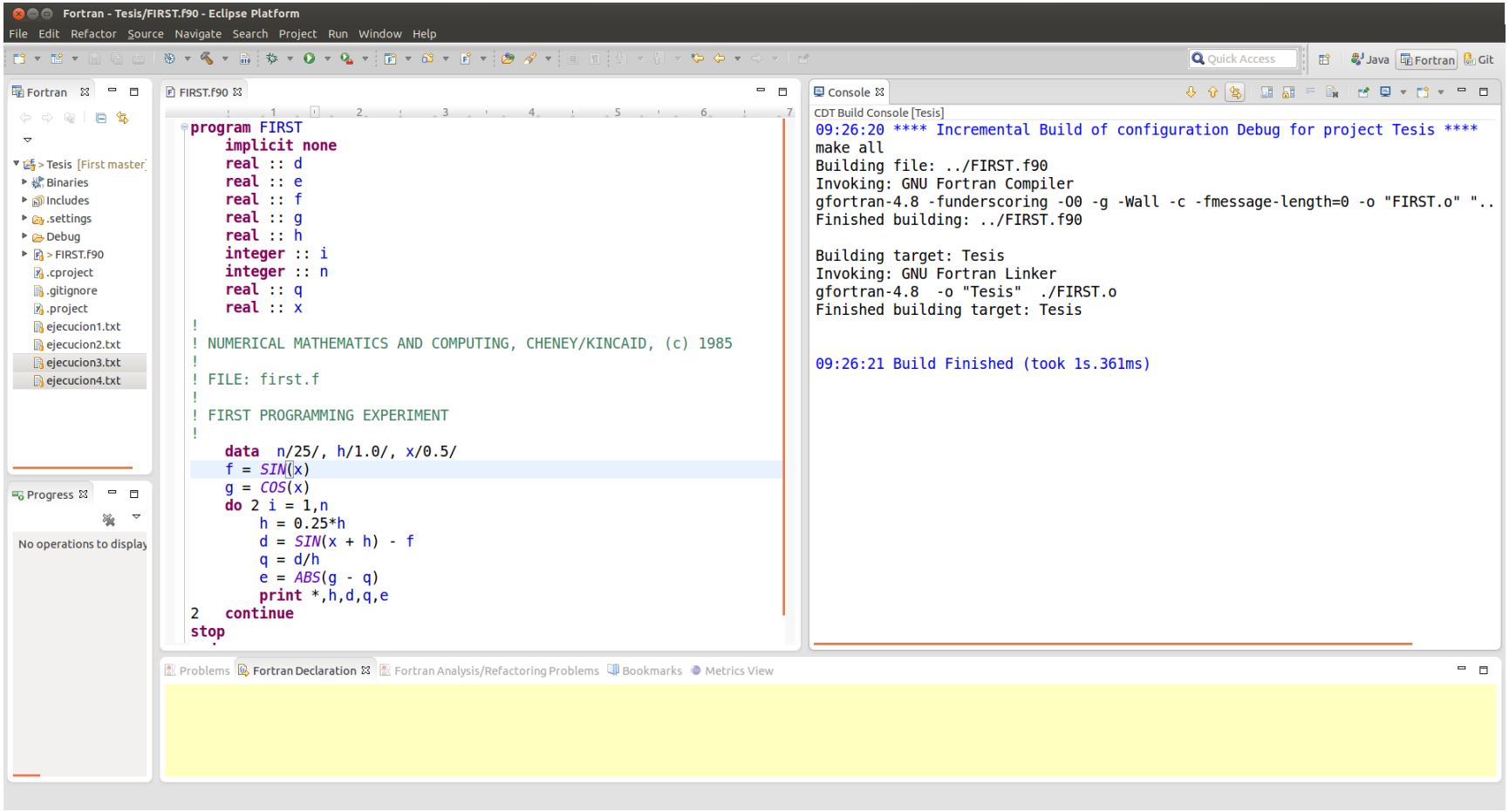


Figura 7.36: Compilación del Código Resultante de la Transformación

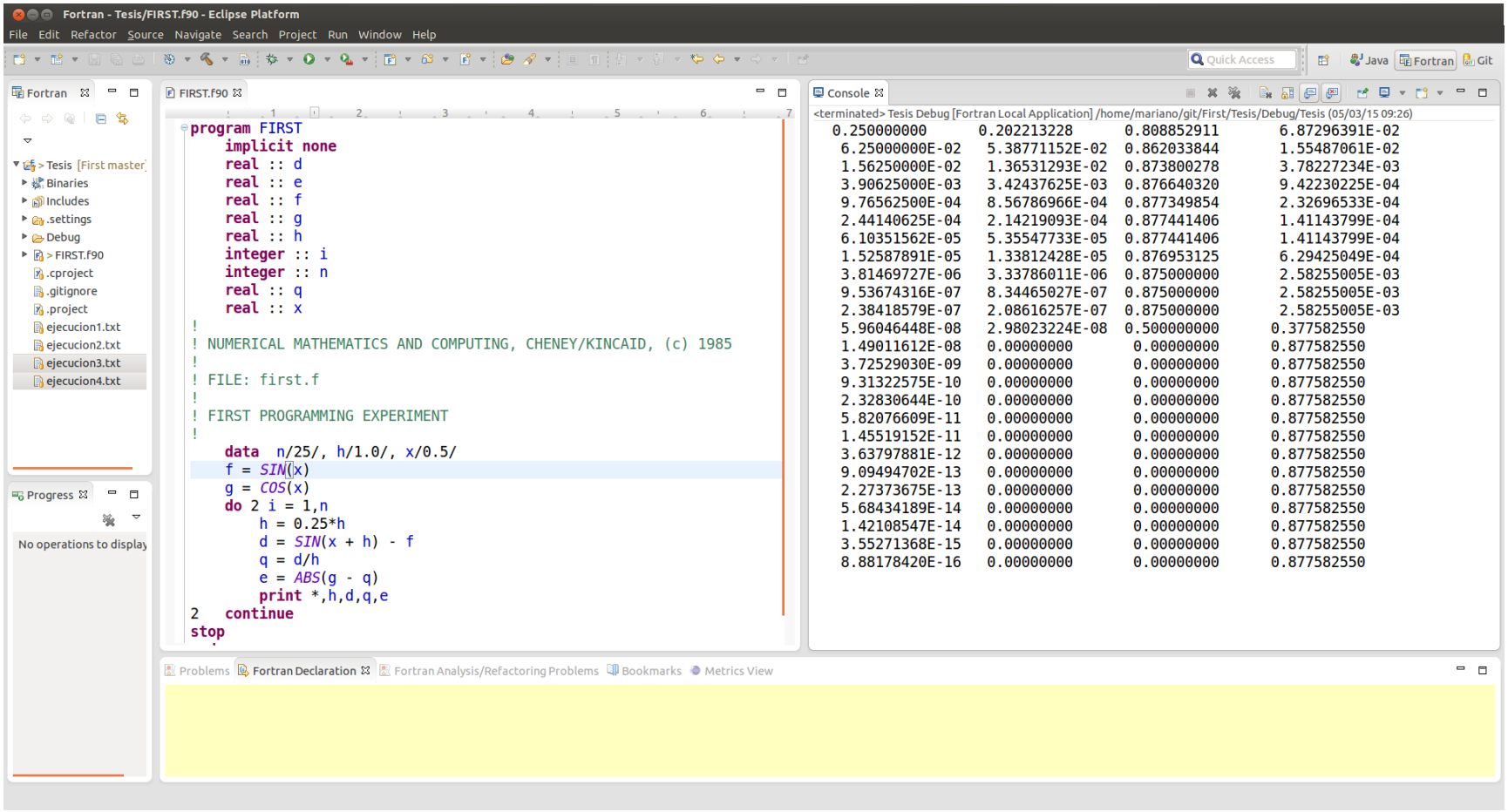


Figura 7.37: Resultados de la Ejecución del Programa

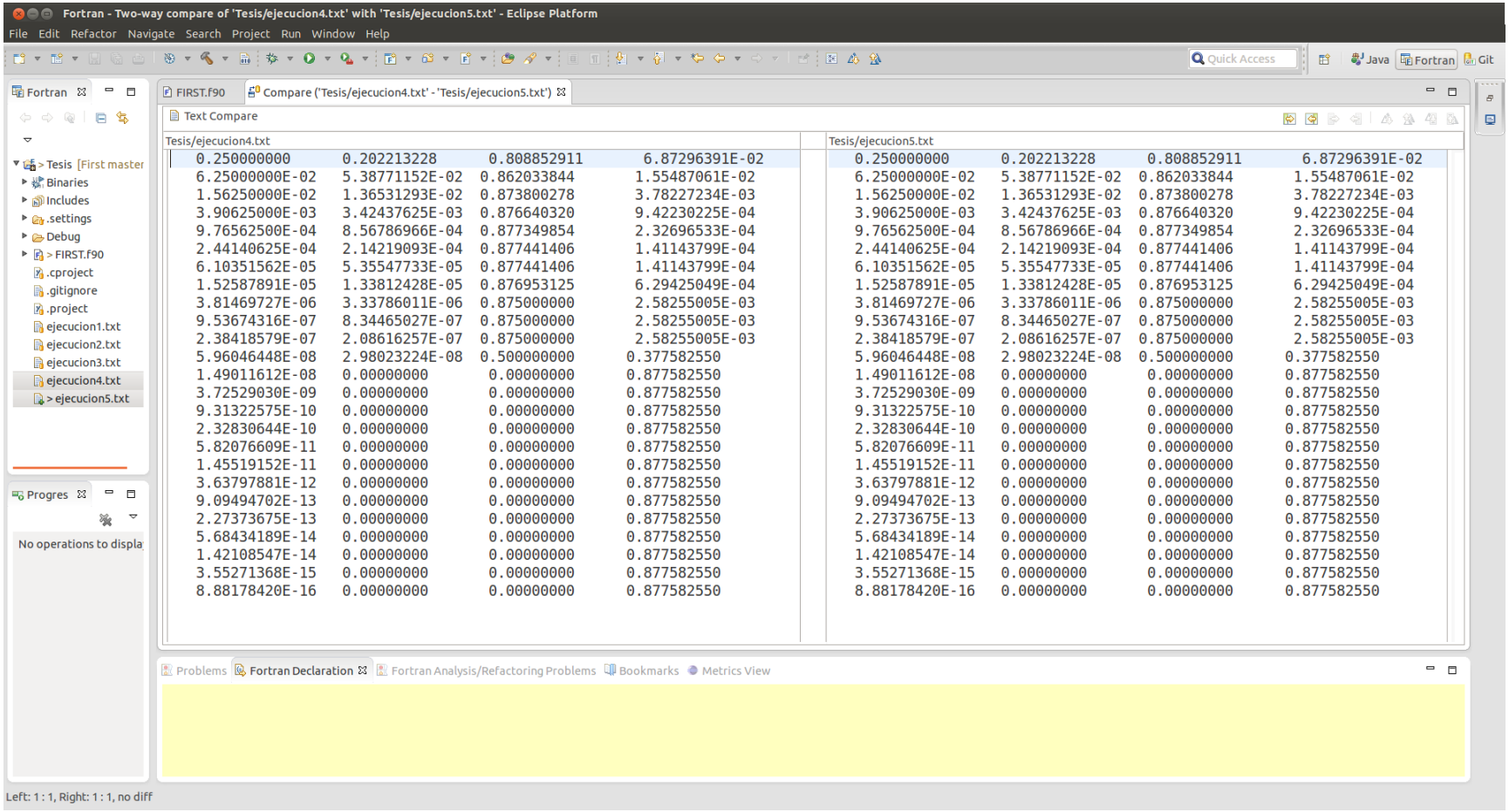


Figura 7.38: Vista de Diferencia (DIFF VIEW) la Comparación de los Resultados de la Ejecución del Código Resultante de la Iteración 3 con el de la Iteración 4

Iteración 5: Reemplazar los ciclos DO escritos de la forma antigua por la moderna

Una de las instrucciones más interesantes desde el punto de vista del cómputo científico es justamente la instrucción DO. En ella reside básicamente el 98 % del cómputo de una aplicación científica. La instrucción DO de Fortran es uno de las objetos más antiguos de la ciencias de la computación, como el lenguaje Fortran en sí mismo. Por ello existen por lo menos 3 formas distintas de escribir el mismo ciclo en Fortran, como se ha visto en el Capítulo 5:

```
....                        ....
DO 110 I=1,10               DO 100 I=1,10
DO 100 J=1,10               DO 100 J=1,10      Loop
   MATRIX(I,J)=0     100     MATRIX(I,J)=0
100 CONTINUE                ....
110 CONTINUE                ....
...                         ....
(A)                          (B)
....
DO I=1,10
 DO J=1,10
   MATRIX(I,J)=0
  END DO
END DO
....
(C)
```

Todas las distintas formas de escribir el ciclo anterior son válidas y compilables. Las versiones (A) y (B) son previas al concepto de programación estructurada. Especialmente la forma (B) llamada Ciclo DO Compartido o Shared DO loop se encuentra en el Apéndice B del estándar de Fortran desde la versión de Fortran 90 [9]. Una de las características más importantes que debería tener el código fuente es que éste tiene que ser fácil de entender y de leer. Sobretodo teniendo en cuenta que el mayor trabajo para paralelizar código fuente es trabajar con él, dado que

la paralelización todavía es un trabajo manual. Para ello como parte de esta línea de investigación se ha implementado una transformación de código fuente que automáticamente cambia los ciclos escritos en los formatos viejos (old style DO loop) por la forma moderna y estructurada [149, 150, 148]. Esta transformación forma parte de la distribución de Photran desde la versión 6.0 año 2010. Al igual que en las otras iteraciones se respeta el flujo de trabajo que puede apreciarse en la Figura 7.38, Figura 7.40, Figura 7.42, Figura 7.43 y Figura 7.44. Cabe destacar que la transformación aplicada ha introducido la instrucción END DO en mayúsculas, por lo tanto como resultado de la fase de retroalimentación de este ciclo se ha decidido agregar una iteración más, la número 11, que deberá volver a dejar a todas las instrucciones del programa escritas en minúsculas. El resultado de la transformación puede apreciarse en la Figura 7.39

```
program FIRST
    implicit none
    real:: d,e,f,g,h,q,x
    integer :: i,n
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
    data  n/25/, h/1.0/, x/0.5/
    f = SIN(x)
    g = COS(x)
    do i = 1,n
        h = 0.25*h
        d = SIN(x + h) - f
        q = d/h
        e = ABS(g - q)
        print *,h,d,q,e
2    continue
   END DO
    stop
end
```

Figura 7.39: Código Fuente del Programa FIRST.f90 Tras Haberse Realizado la Trasformación de la Iteración 5

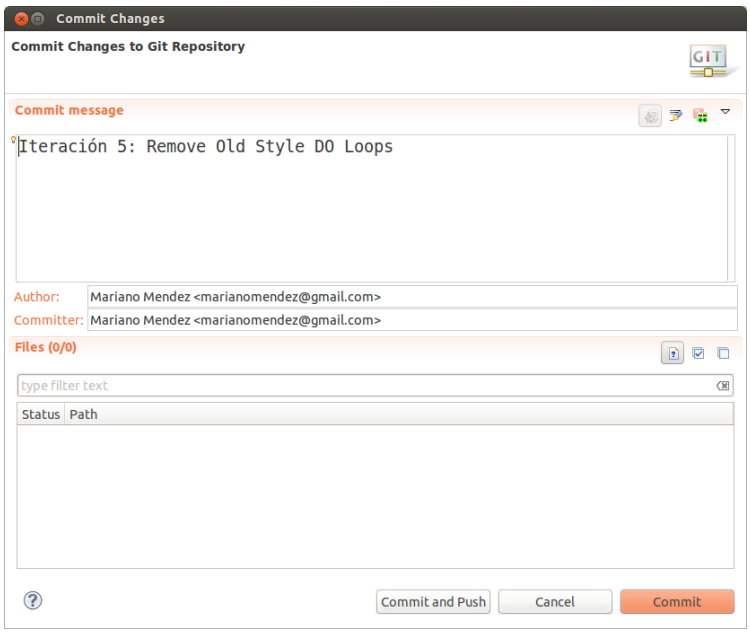


Figura 7.40: Commit de la Versión de Trabajo

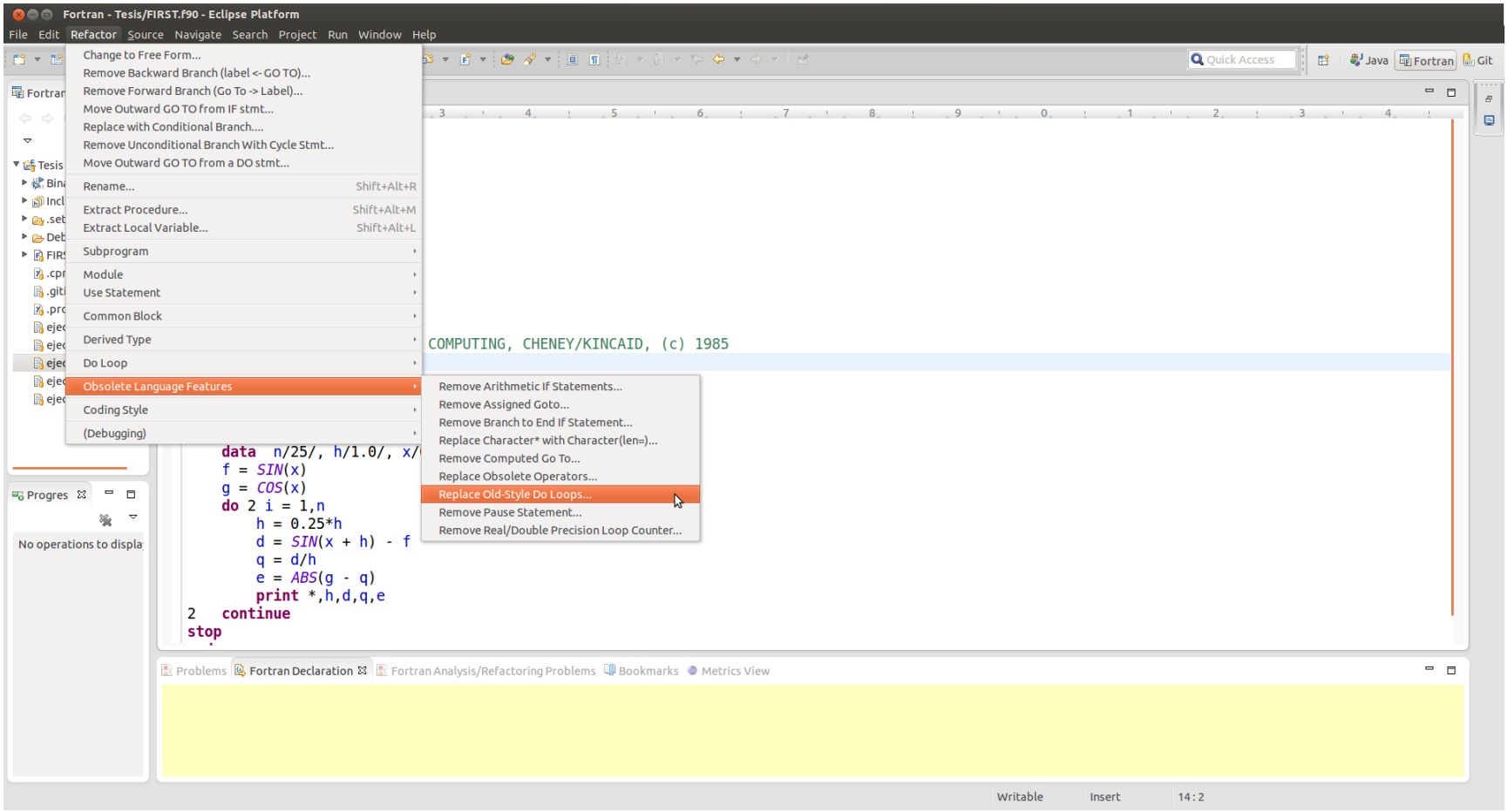


Figura 7.41: Selección de la Transformación Replace Old Style Do Loops en el Menú de la Aplicación

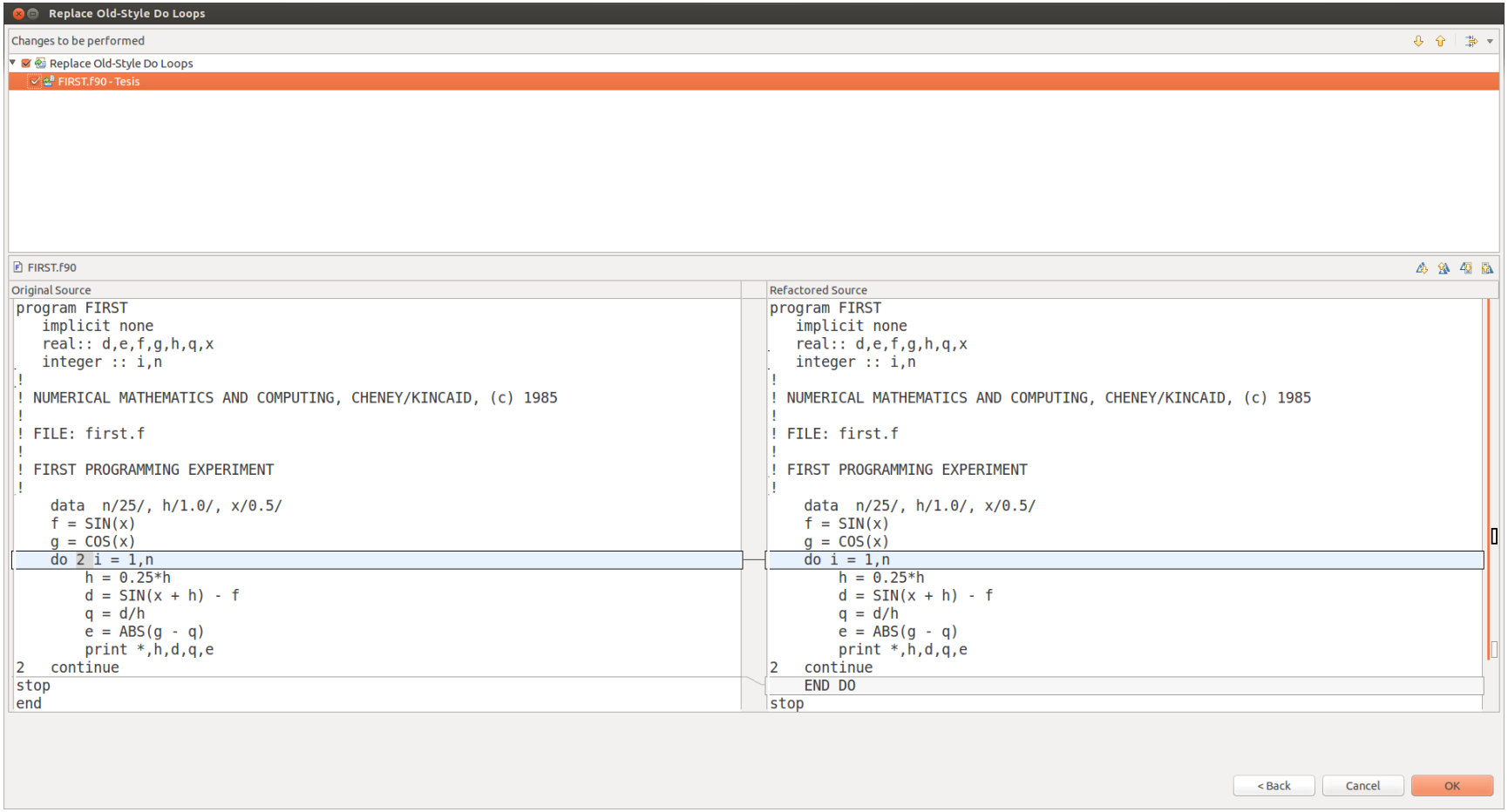


Figura 7.42: Vista Previa en Diferencia (DIFF VIEW) del Código Resultante de la Transformación de la Iteración 5

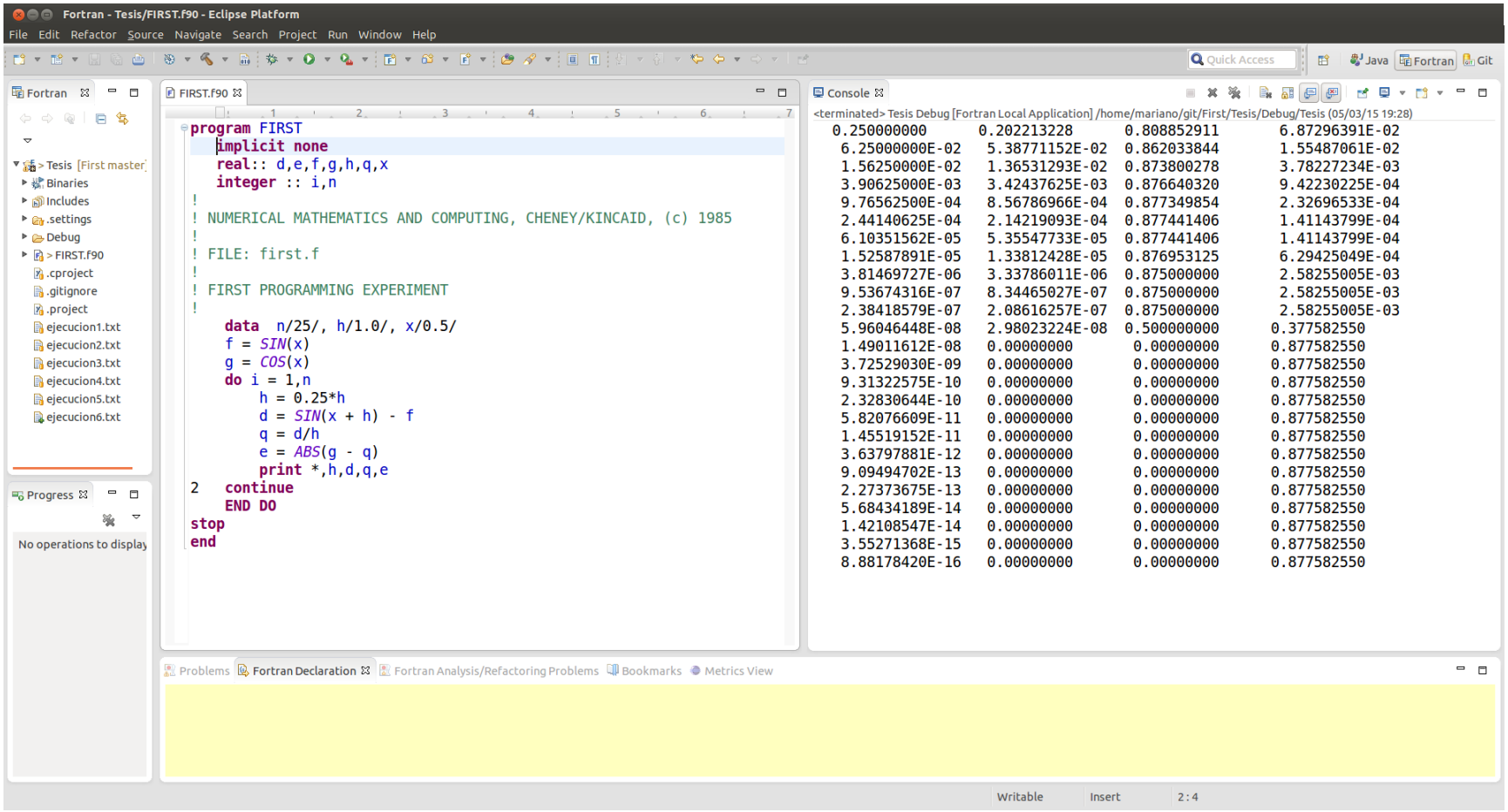


Figura 7.43: Resultados de la Corrida del Programa FIRST.f90 (Iteración 5)

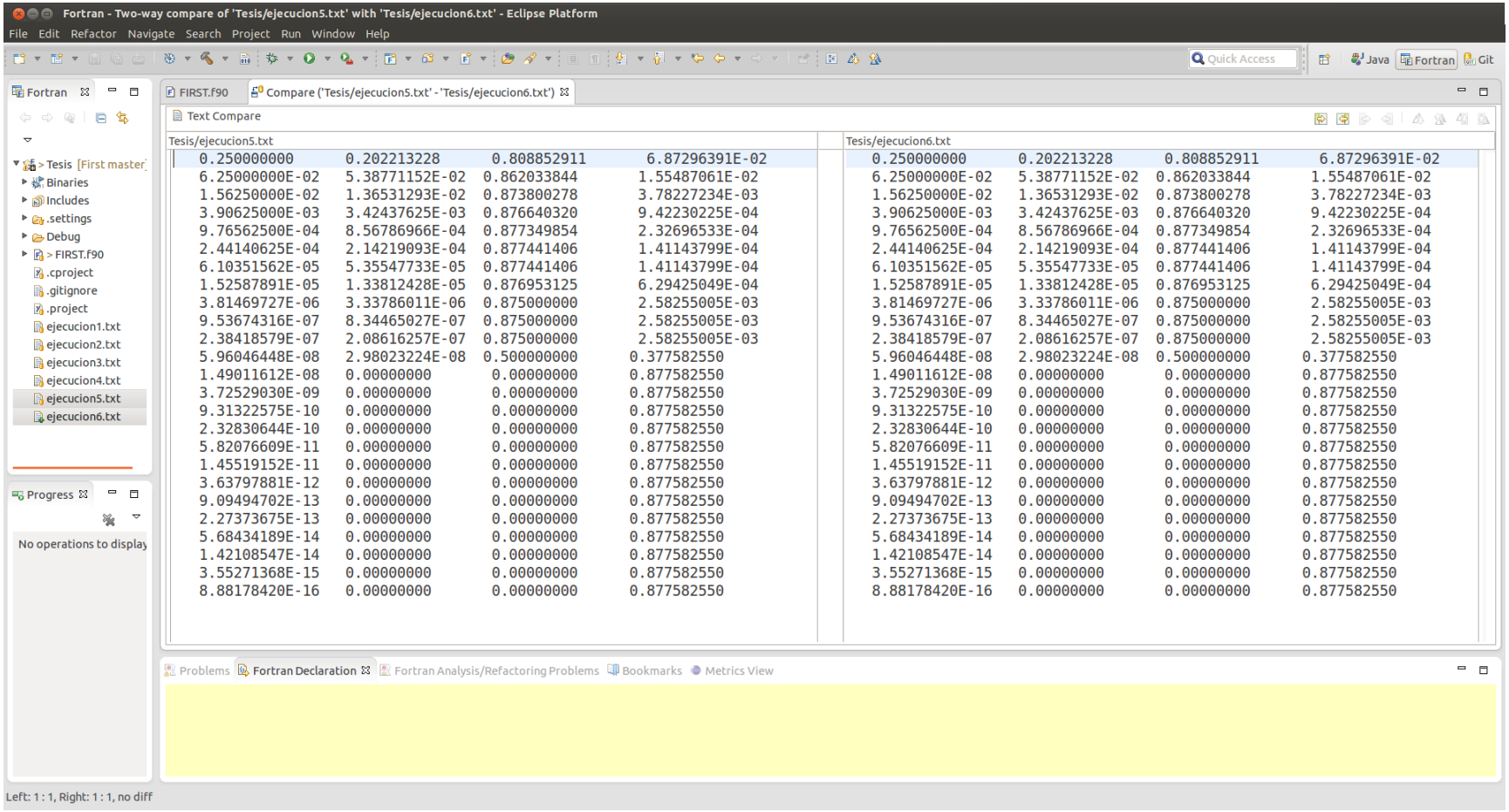


Figura 7.44: Vista de Diferencia (DIFF VIEW) la Comparación de los Resultados de la Ejecución del Código Resultante de la Iteración 4 con el de la Iteración 5

Iteración 6: Eliminar etiquetas no utilizadas

En este punto tras haber eliminado las instrucciones DO escritas en formato viejo, las etiquetas de las instrucciones CONTINUE o de las instrucciones de terminación compartida, muy posiblemente no sean más referenciadas dentro del programa. Para ello se ha implementado una transformación que elimina las etiquetas que ya no están siendo referenciadas. Esta transformación forma parte de la distribución oficial de la herramienta Photran y ha sido implementada como parte de esta línea de investigación. Como se ha realizado en otras iteraciones se fija la versión de la iteración 5 como código fuente de partida. Se sigue el flujo de trabajo y se lleva a cambio la transformación y su posterior verificación. Ver Figura 7.44, Figura 7.46, Figura 7.47, Figura 7.48, Figura 7.49 y Figura 7.50. El resultado de la transformación puede apreciarse en la Figura 7.45

```
 program FIRST
      implicit none
      real:: d,e,f,g,h,q,x
      integer :: i,n
 !
 ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
 !
 ! FILE: first.f
 !
 ! FIRST PROGRAMMING EXPERIMENT
 !
      data  n/25/, h/1.0/, x/0.5/
      f = SIN(x)
      g = COS(x)
      do i = 1,n
         h = 0.25*h
         d = SIN(x + h) - f
         q = d/h
         e = ABS(g - q)
         print *,h,d,q,e
        continue
     END DO
     stop
end
```

Figura 7.45: Código Fuente del Programa FIRST.f90 Tras Haber Finalizado la Iteración 6

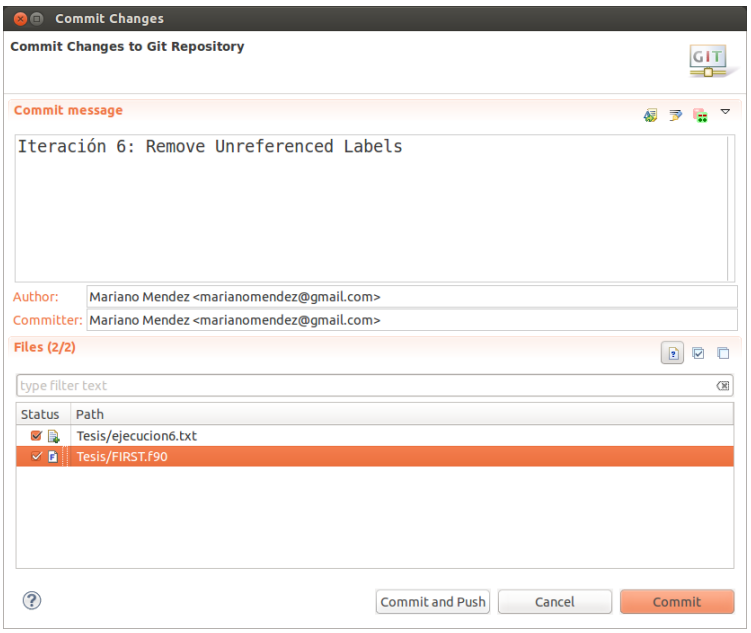


Figura 7.46: Commit de la Versión de Trabajo

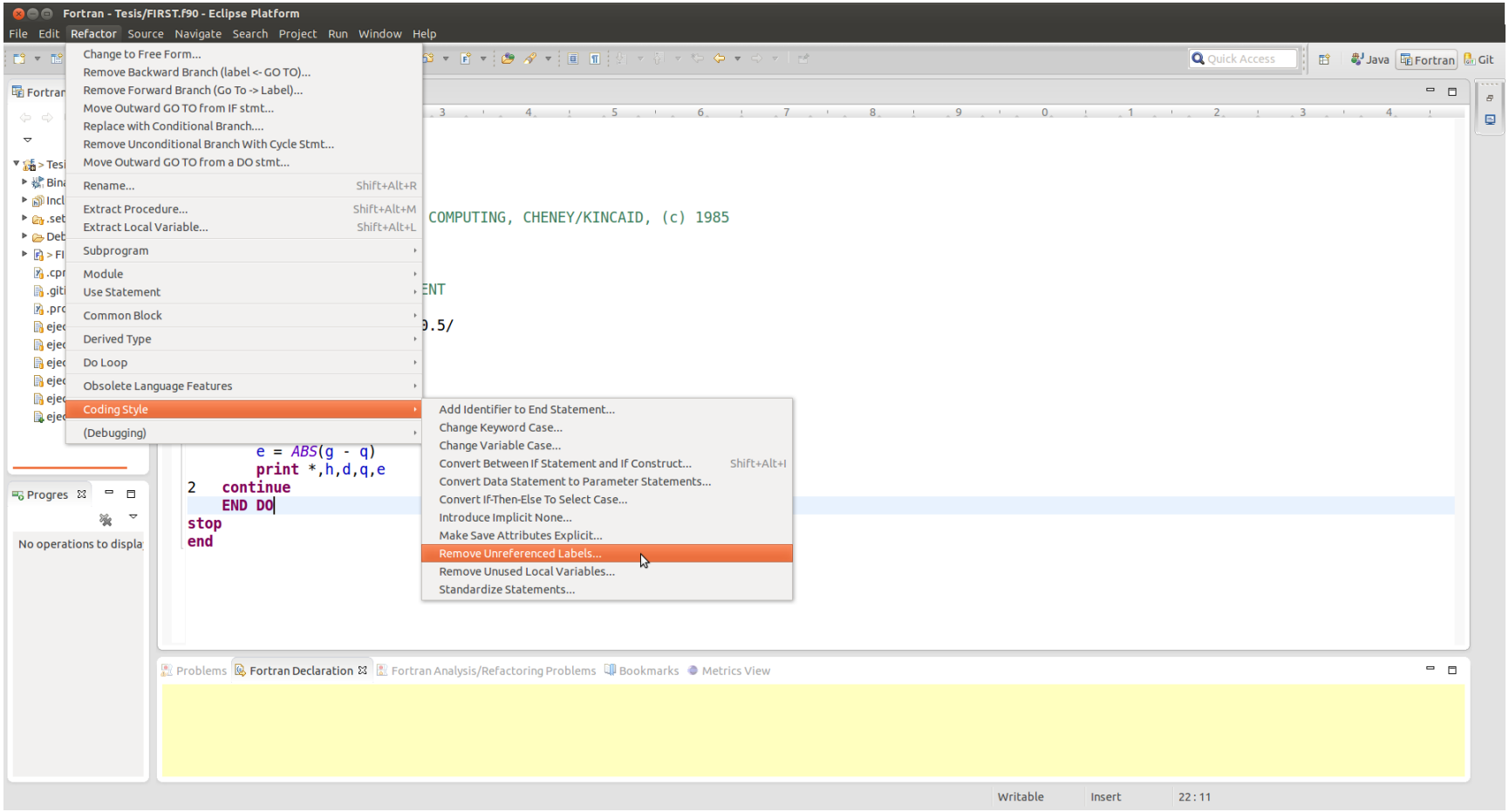


Figura 7.47: Selección de la Opción de Menú de la Transformación que Elimina Etiquetas no Referenciadas.

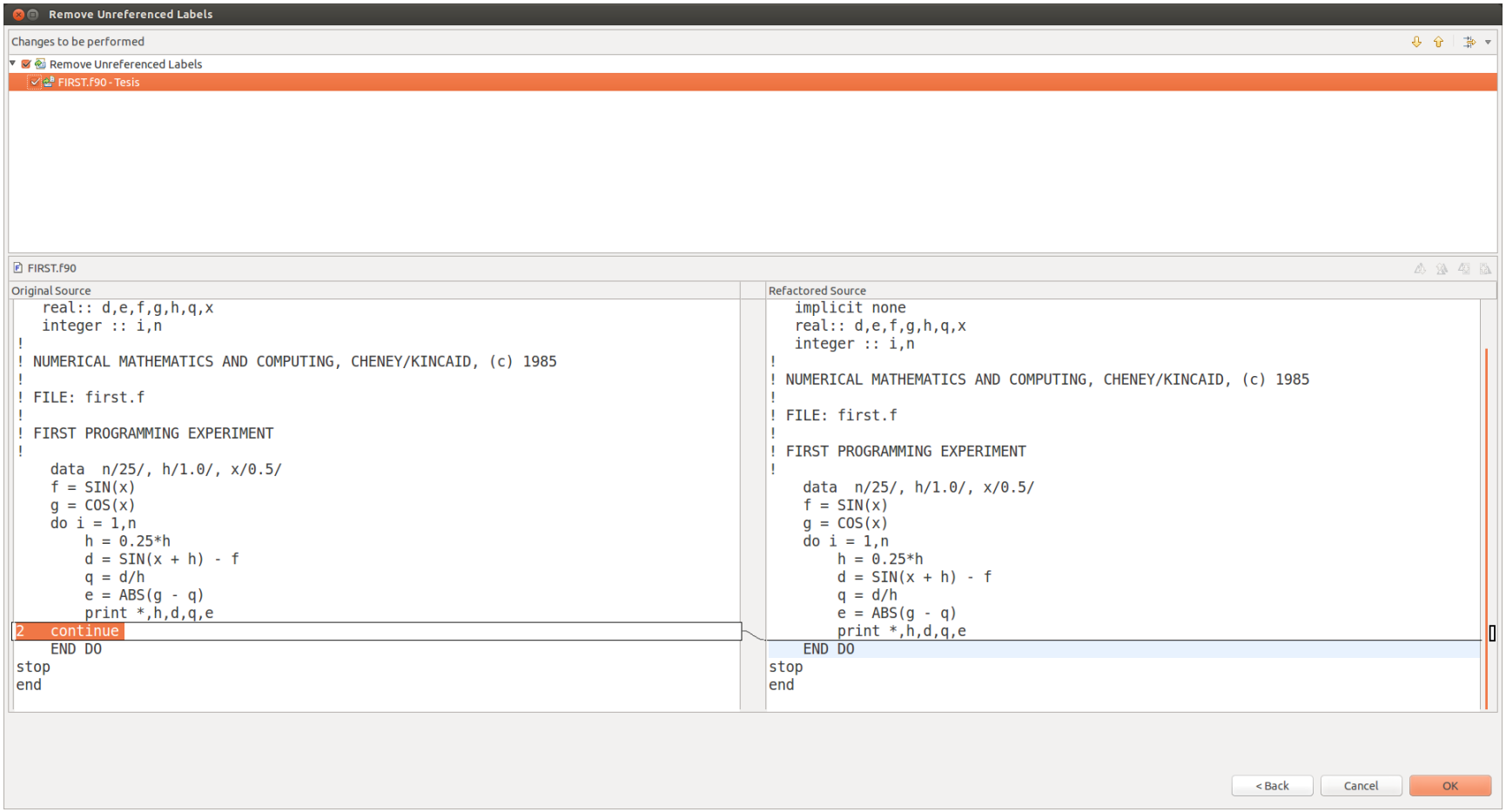


Figura 7.48: Vista Previa en Diferencia (DIFF VIEW) del Código Resultante de la Rransformación de la Iteración 5

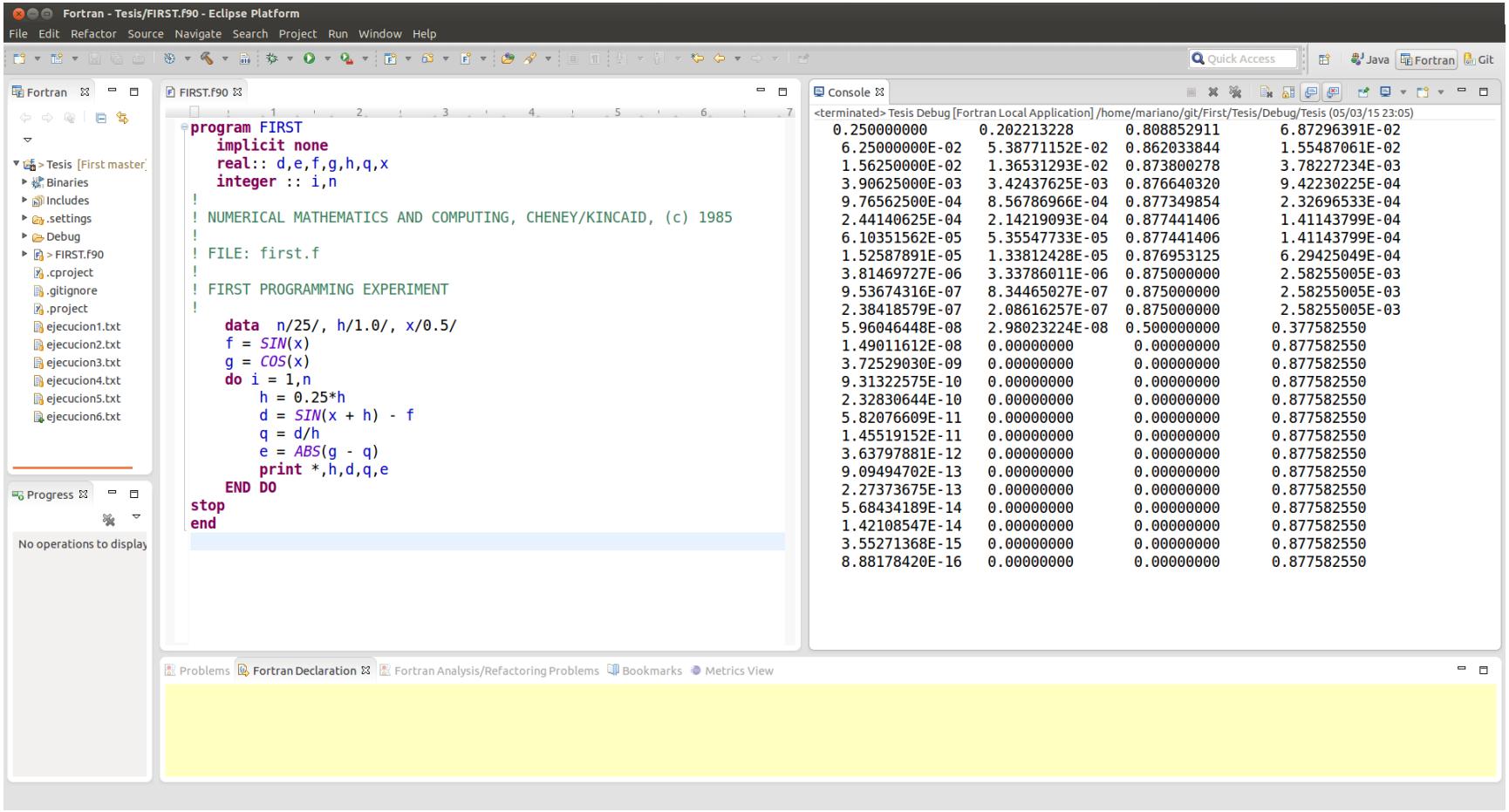


Figura 7.49: Resultados de la Corrida del Programa FIRST.f90 (Iteración 5)

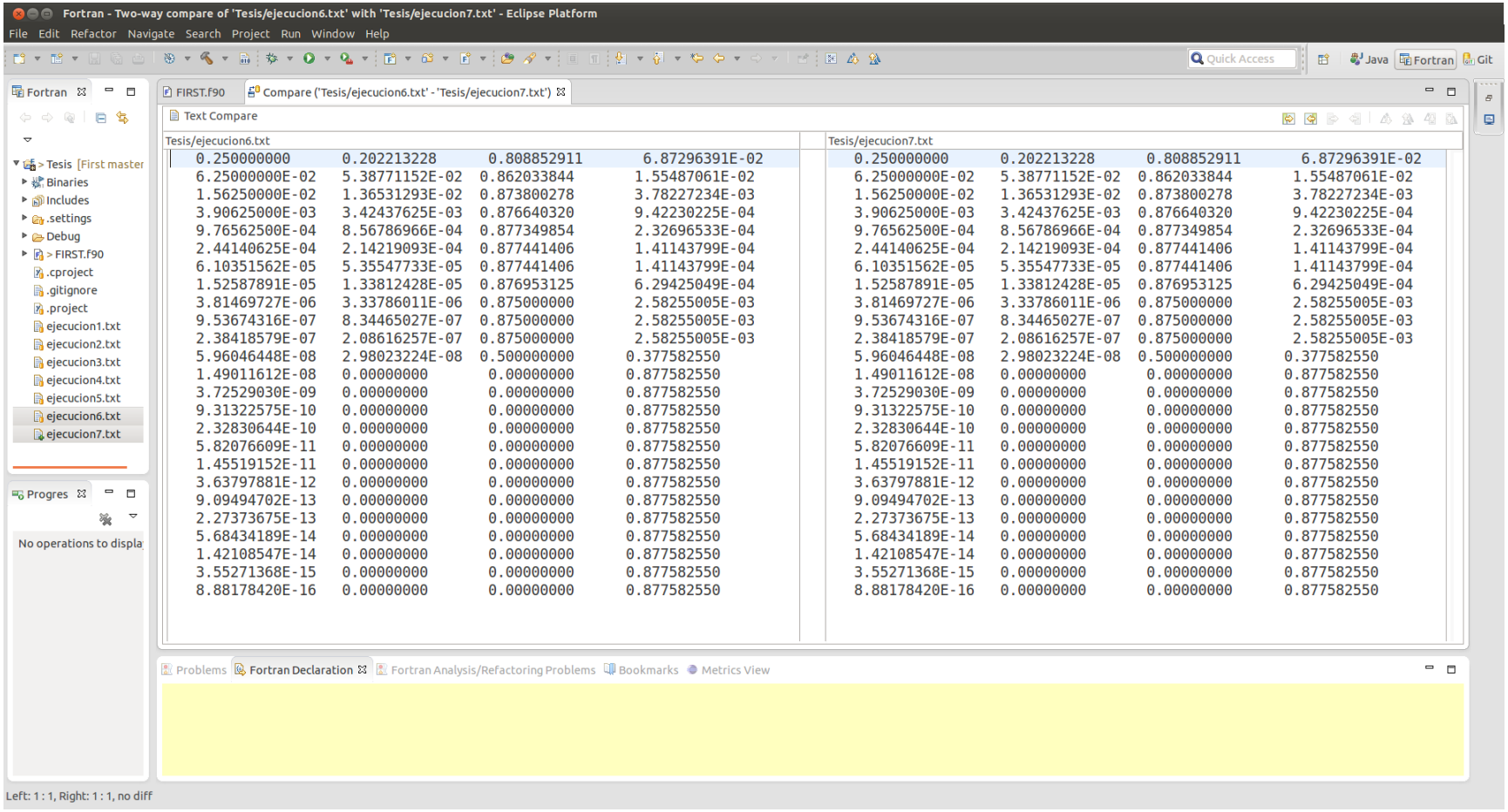


Figura 7.50: Vista de Diferencia (DIFF VIEW) la Comparación de los Resultados de la Corrida del Código Resultante de la Iteración 5 con el de la Iteración 6

Iteración 7: Eliminar la instrucciones no requeridas (*)

En este caso se eliminarán las instrucciones CONTINUE y la instrucción STOP. Ninguna aporta funcionalidad del programa. La primera instrucción según el estándar de FORTRAN 66: "la ejecución de esta secuencia causa la continuación normal de la secuencia" Sección 7.1.2.6 [87], en otras palabras equivale a una instrucción No Operar. La segunda instrucción, el STOP, fue utilizado por FORTRAN 66 y FORTRAN 77 para ser la última instrucción ejecutable en el programa dado que el END no fue una instrucción ejecutable hasta la edición del estándar de Fortran 90 [87, 8, 9]. Esta transformación ha sido identificada en la etapa de comprensión a lo largo de las iteraciones, y es una buena candidata a ser implementada para contribuir con la herramienta. Al día de la fecha no se encuentra implementada, por lo tanto esta transformación ha sido realizada en forma manual, pero siguiendo el flujo de trabajo planteado en el desarrollo guiado por el cambio. Ver Figura 7.52, Figura 7.53 y Figura 7.54. El resultado de la transformación puede apreciarse en la Figura 7.45

```
program FIRST
      implicit none
      real:: d,e,f,g,h,q,x
      integer :: i,n
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
     data n/25/, h/1.0/, x/0.5/
     f = SIN(x)
     g = COS(x)
     do i = 1,n
        h = 0.25*h
        d = SIN(x + h) - f
        q = d/h
        e = ABS(g - q)
        print *,h,d,q,e
     END DO
end
```

Figura 7.51: Código Fuente del Programa FIRST.f90 Resultado de la Iteración 7

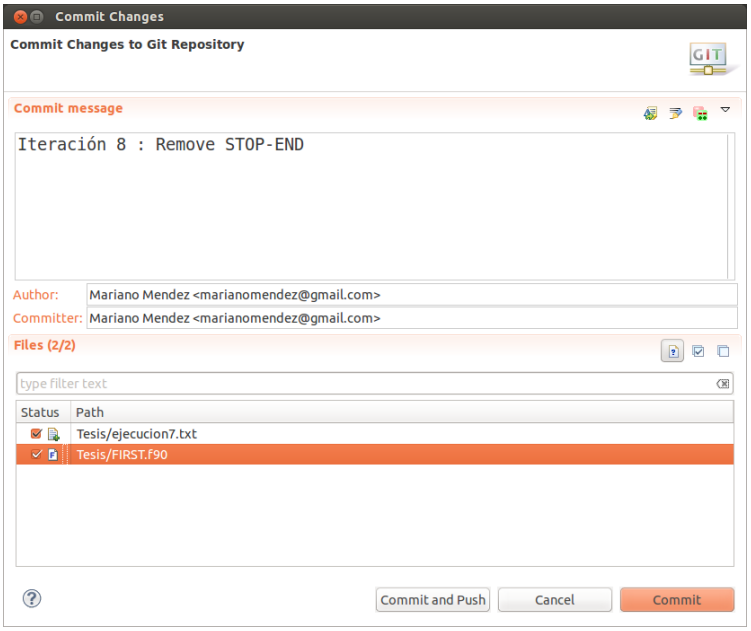


Figura 7.52: Commit de la Versión de Trabajo

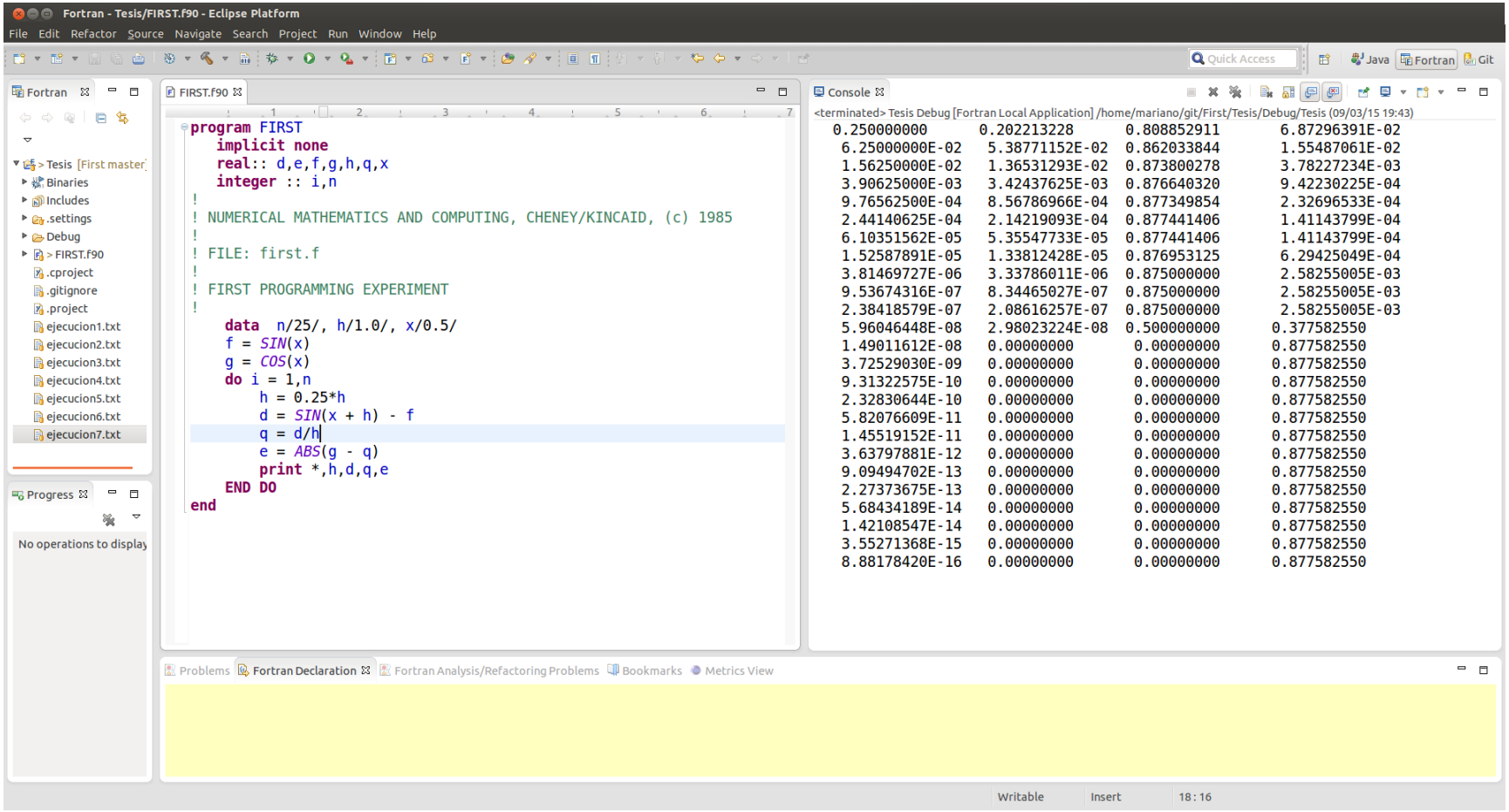


Figura 7.53: Resultados de la Corrida del Programa FIRST.f90 (Iteración 7)

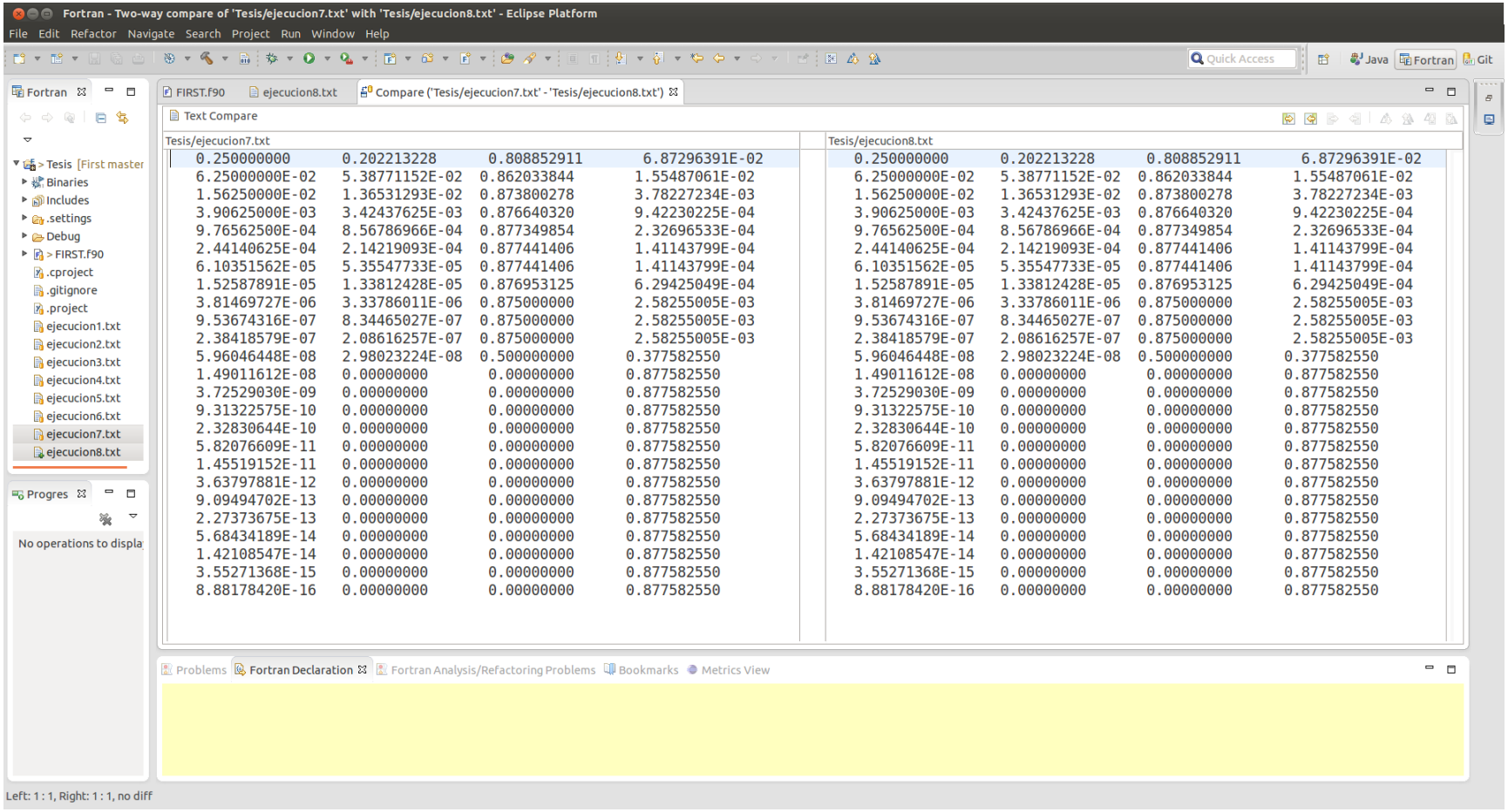


Figura 7.54: Vista de Diferencia (DIFF VIEW) de la Comparación de los Resultados de la Corrida del Código Resultante de la Iteración 6 con el de la Iteración 7

Iteración 8: Agregar a la instrucción END PROGRAM el nombre del programa

Como se ha visto en la iteración anterior la instrucción END de FORTRAN 77 y FORTRAN 66 no fue considerada ejecutable hasta el estándar de Fortran 90, Sección 2.3.3. Además no existían las combinaciones END-PROGRAM, END-FUNCTION y END-SUBROUTINE. Por ello una transformación que actualiza esta característica antigua del lenguaje agrega a la instrucción END la palabra PROGRAM y seguidamente el nombre del programa, ver Sección 2.3.3 del estándar de Fortran 90, ver Figura 7.55, Figura 7.56, Figura 7.57, Figura 7.58 y Figura 7.59.

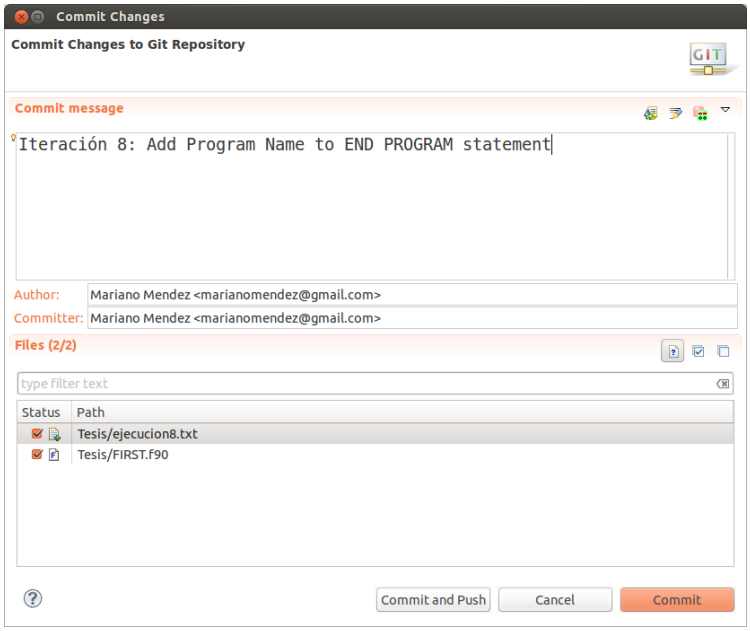


Figura 7.55: Commit de la Versión de Trabajo

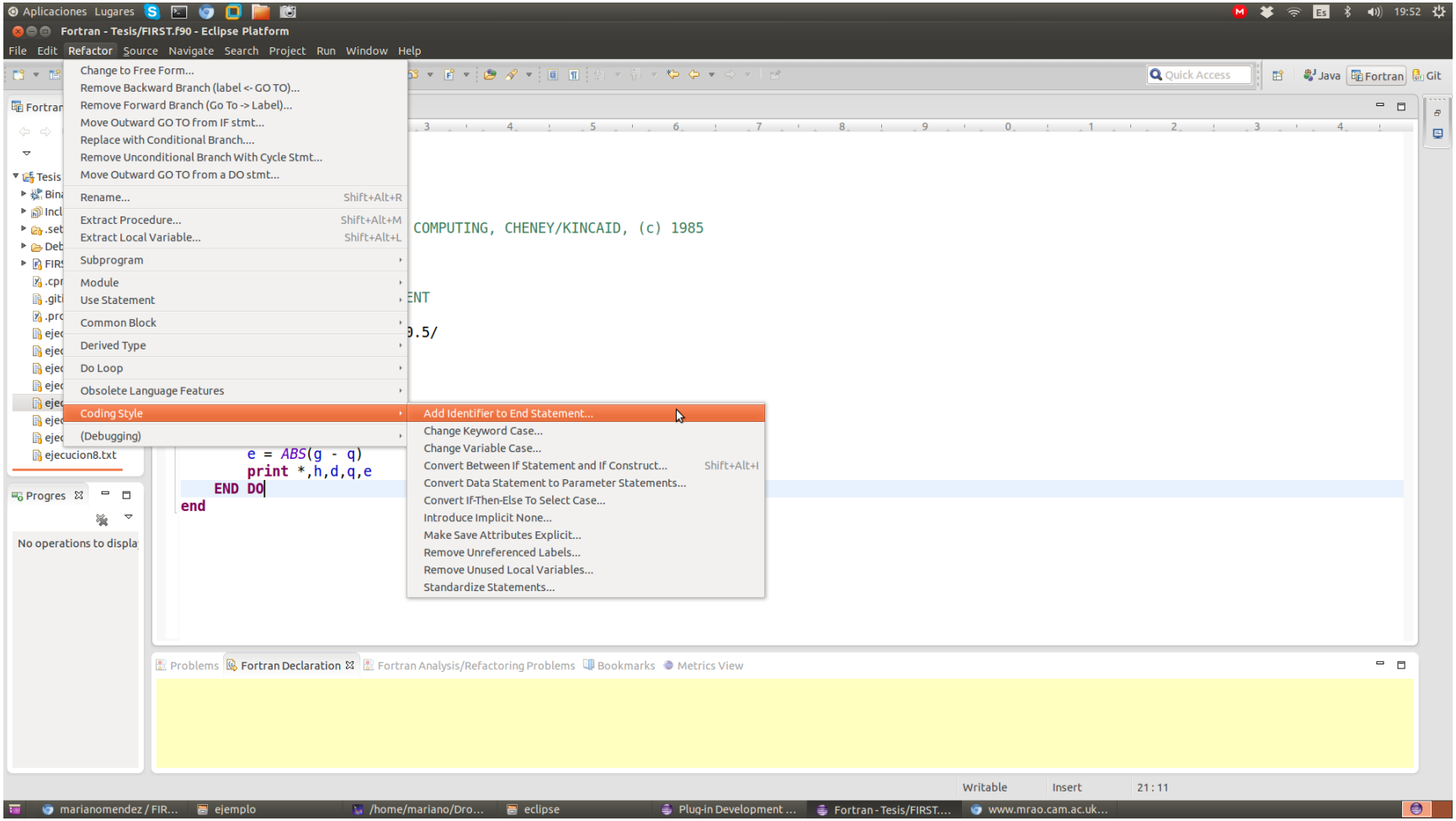


Figura 7.56: Selección de la Opción de Menú de la Transformación que Agega el Correspondiente Identificador a la Instrucción END.

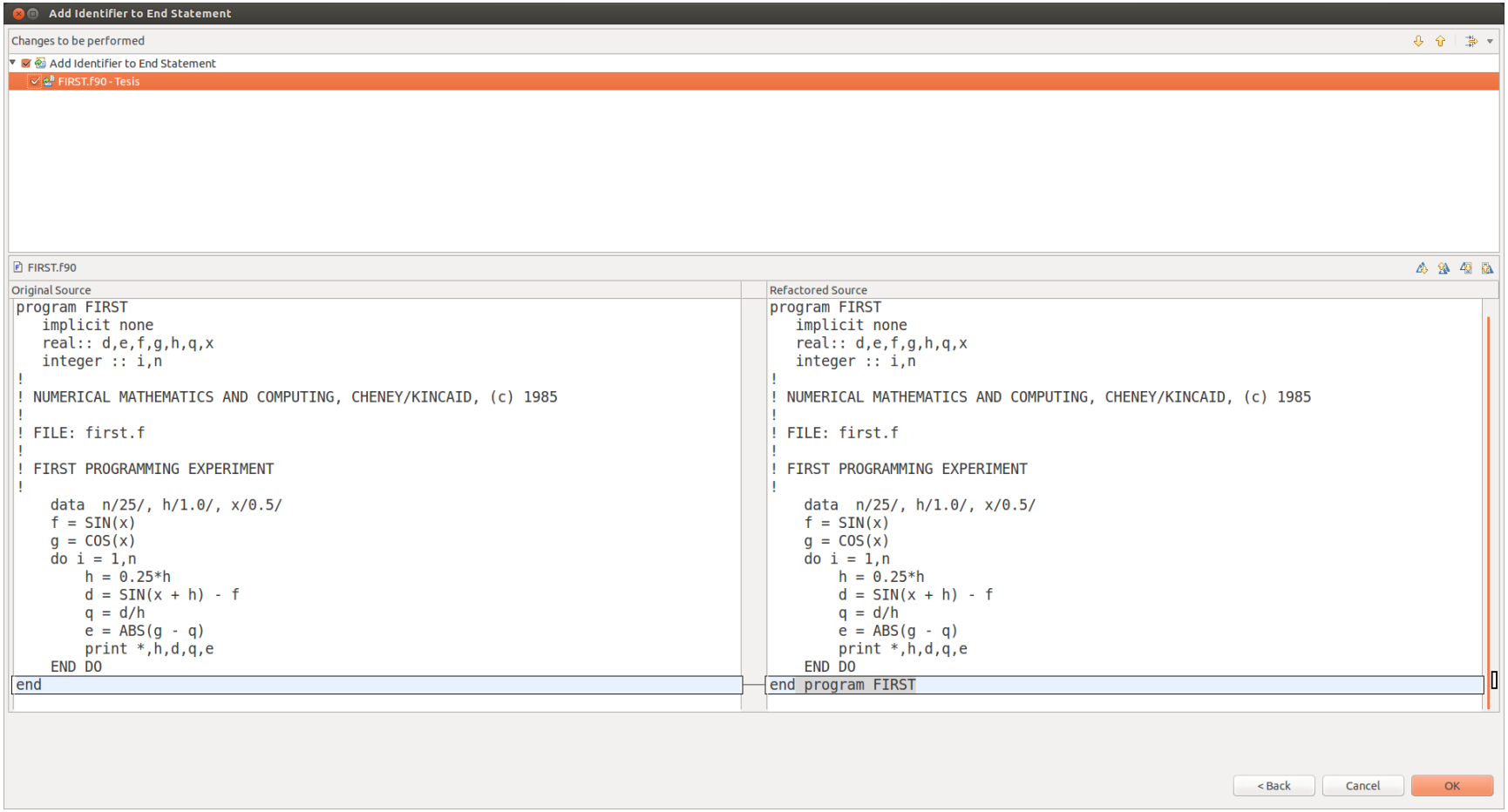


Figura 7.57: Vista Previa en diferencia (DIFF VIEW) del Código Resultante de la Transformación de la Iteración 8

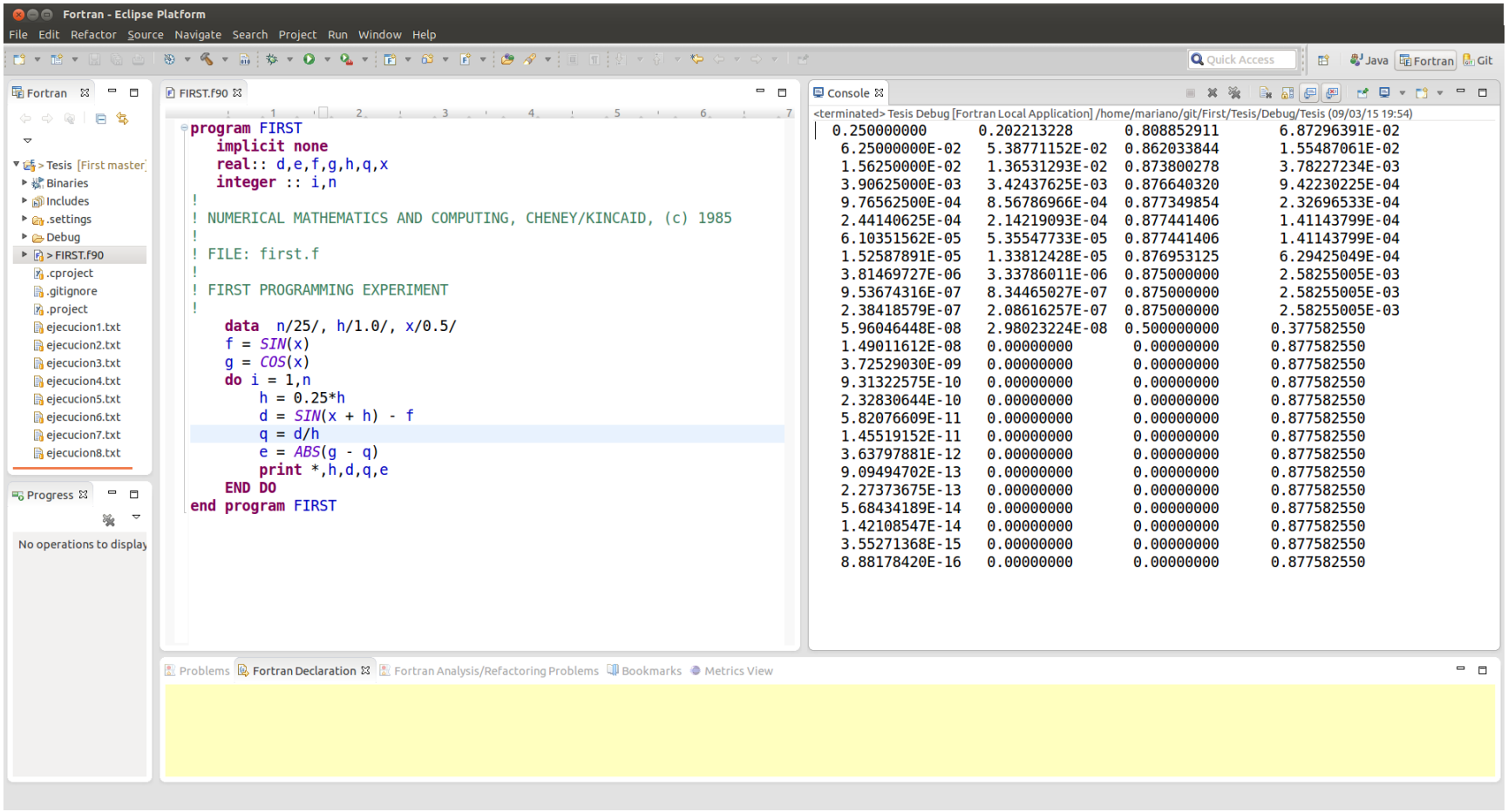
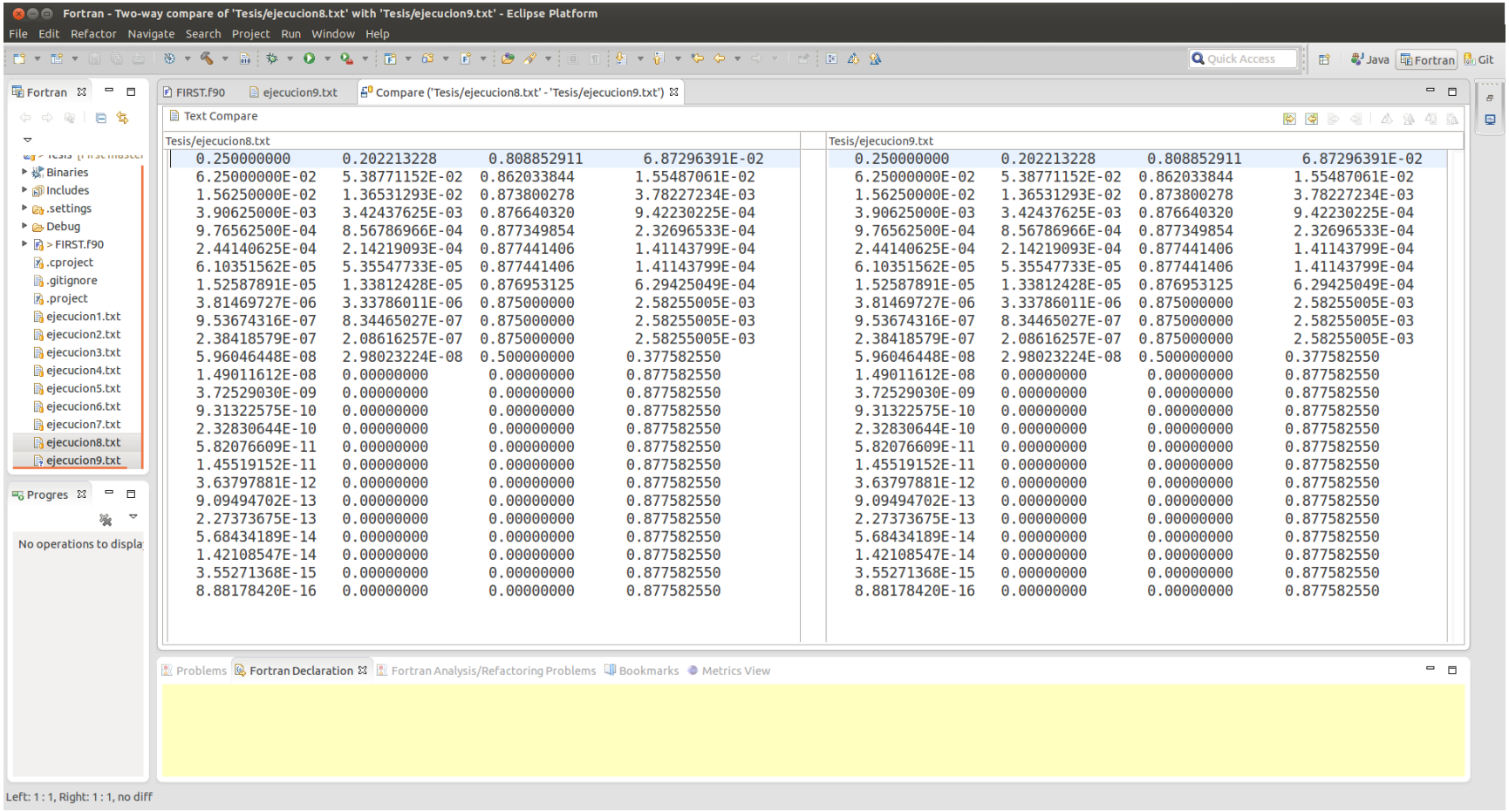


Figura 7.58: Resultados de la Corrida del Programa FIRST.f90 (Iteración 8)



0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

Figura 7.59: Vista de Diferencia (DIFF VIEW) de la Comparación de los Resultados de la Corrida del Código Resultante de la Iteración 7 con el de la Iteración 8

Iteración 9: Reemplazar el bloque DATA por las correspondientes asignaciones variables (*)

La inicialización de una variable en una declaración de tipo o de cualquier parte de la variable en una instrucción DATA implica que la variable tiene el atributo SAVE a menos que la ésta esté en un COMMON BLOCK con nombre. Según la Sección 5.1.2.5 del Estándar de Fortran 90 "el atributo SAVE especifica que los objetos declarados en una declaración que contenga este atributo conservan su estado de asociación, estado de asignación, estado definición, y el valor después de la ejecución de una sentencia RETURN o END en la unidad de alcance que contiene la declaración" [9], esto es el equivalente del atributo **static** de C. En el caso de este programa las variables n y h no tienen ningún motivo claramente definido para tener el atributo SAVE, es decir, no tienen por qué ser variables estáticas, por ello deberían estar inicializadas en una asignación, ver Figura 7.61, Figura 7.62 y Figura 7.63. Esta transformación es una nueva transformación propuesta en este trabajo, dado que aun no ha sido implementada. El resultado de la transformación puede apreciarse en la Figura 7.60

```
program FIRST
     implicit none
     real:: d,e,f,g,h,q,x
     integer :: i,n
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
     data  n /25/
     x=0.5
     h=1.0
     f = SIN(x)
     g = COS(x)
     do i = 1,n
        h = 0.25*h
        d = SIN(x + h) - f
        q = d/h
        e = ABS(g - q)
        print *,h,d,q,e
     END DO
end
```

Figura 7.60: Código Fuente del Programa FIRST.f90 Resultado de la Iteración 9

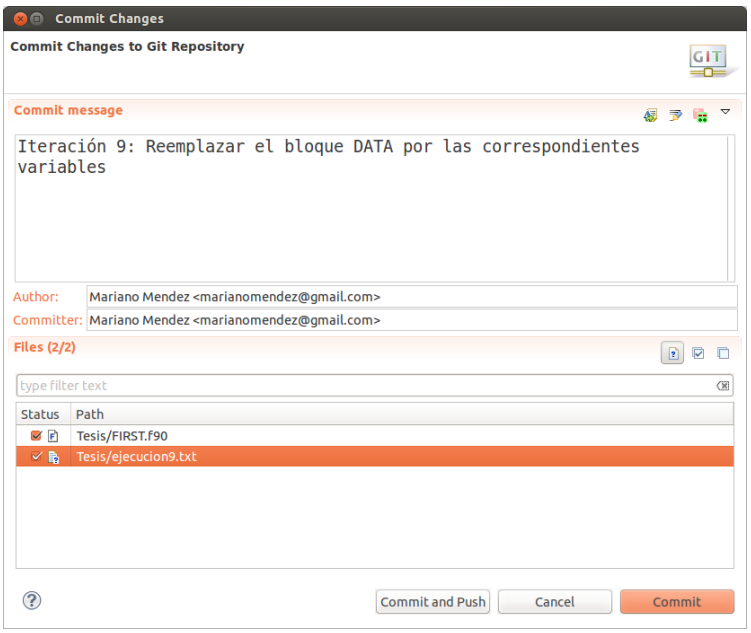


Figura 7.61: Commit de la Versión de Trabajo

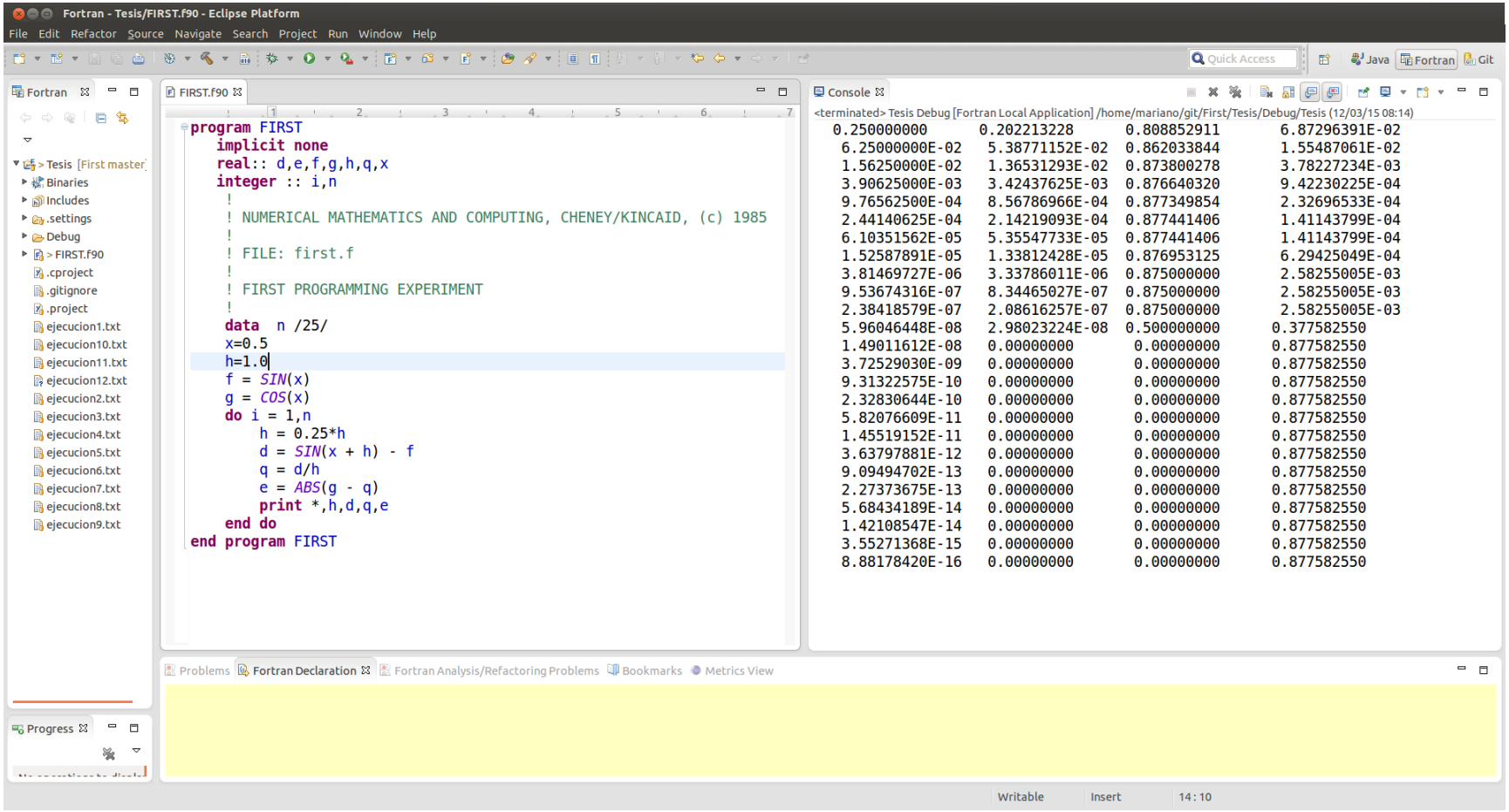


Figura 7.62: Resultados de la Corrida del Programa FIRST.f90 (Iteración 9)

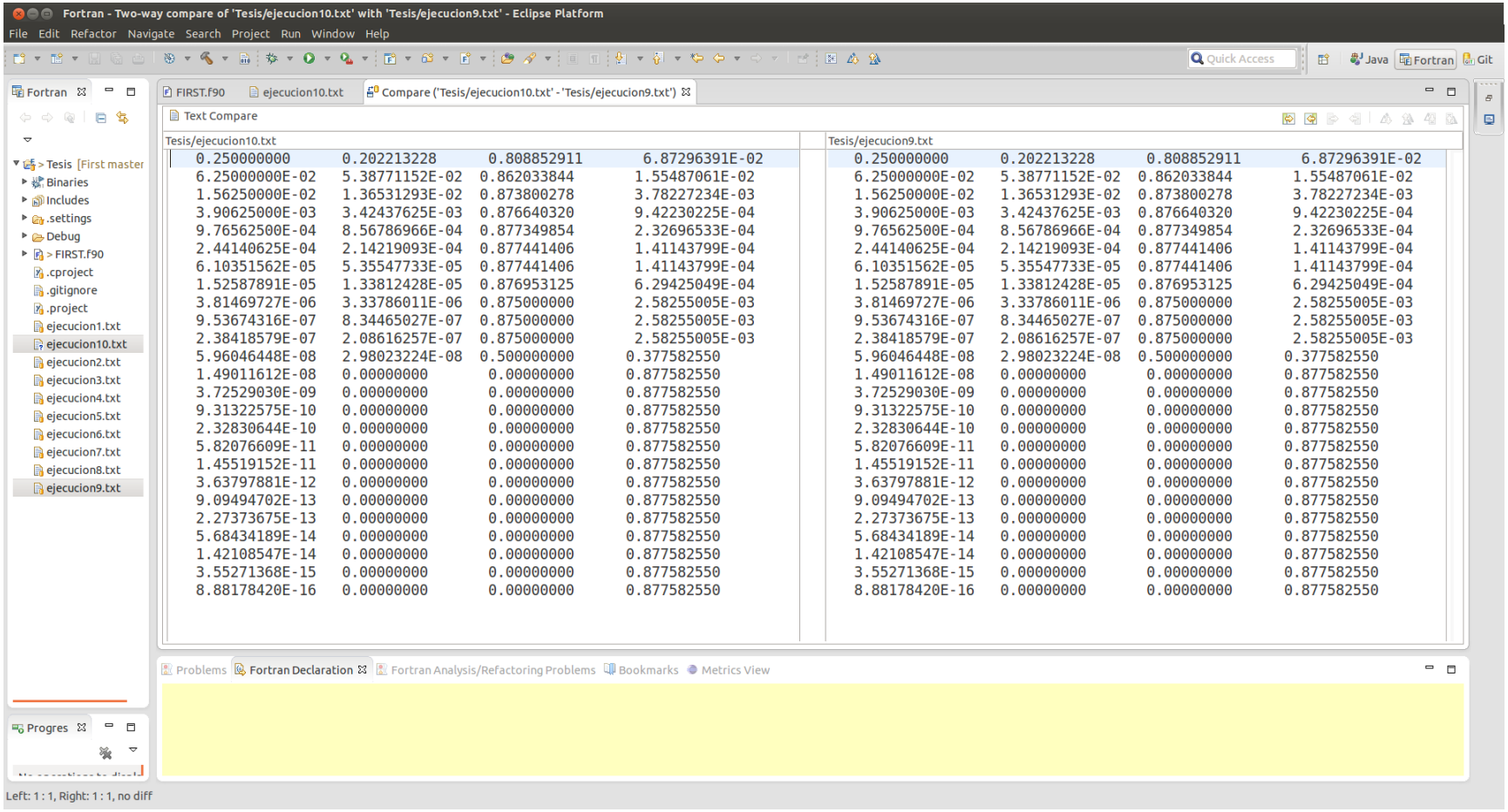


Figura 7.63: Vista de Diferencia (DIFF VIEW) de la Comparación de los Resultados de la Ejecución del Código Resultante de la Iteración 8 con el de la Iteración 9

Iteración 10: Reemplazar el bloque DATA por las correspondientes Constantes

Esta transformación tiene como objeto extraer constantes de las instrucciones DATA. En este caso la constante es la variable n, que representa al número de iteraciones que se deben realizar para cumplir con el objetivo del programa. Para ello se utiliza una transformación ya implementada en la herramienta que justamente tiene ese propósito. Siempre utilizando el flujo de trabajo propuesto se procede a realizar la transformación/cambio. Ver Figura 7.65, Figura 7.66, Figura 7.67, Figura 7.68 y Figura 7.69. El resultado de la transformación puede apreciarse en la Figura 7.64.

```
program FIRST
     implicit none
     real:: d,e,f,g,h,q,x
     integer :: i,n
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
     parameter ( n = 25 )
     x=0.5
     h=1.0
     f = SIN(x)
     g = COS(x)
     do i = 1,n
        h = 0.25*h
        d = SIN(x + h) - f
        q = d/h
        e = ABS(g - q)
        print *,h,d,q,e
     END DO
end program FIRST
```

Figura 7.64: Código Fuente del Programa FIRST.f90

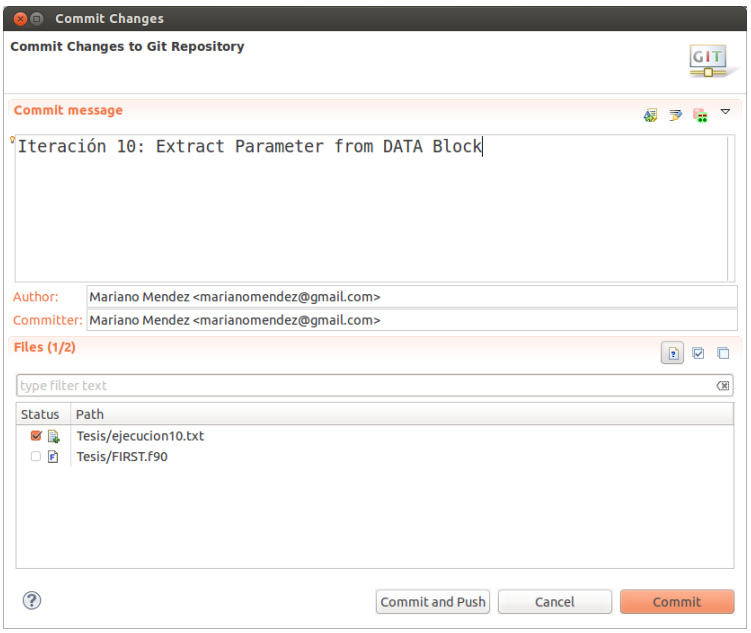


Figura 7.65: Commit de la Versión de Trabajo

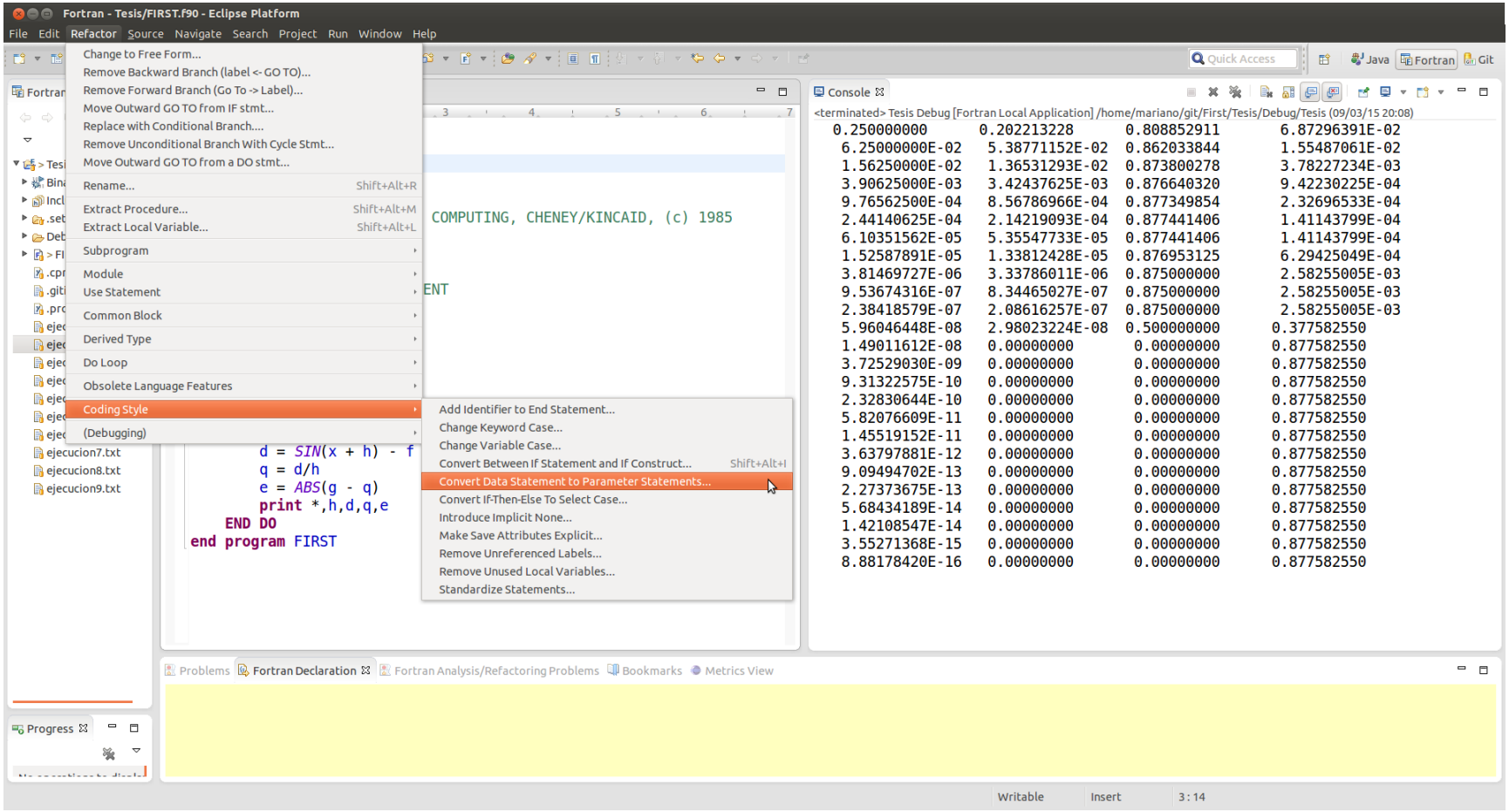


Figura 7.66: Selección de la Opción de Menú de la Transformación.

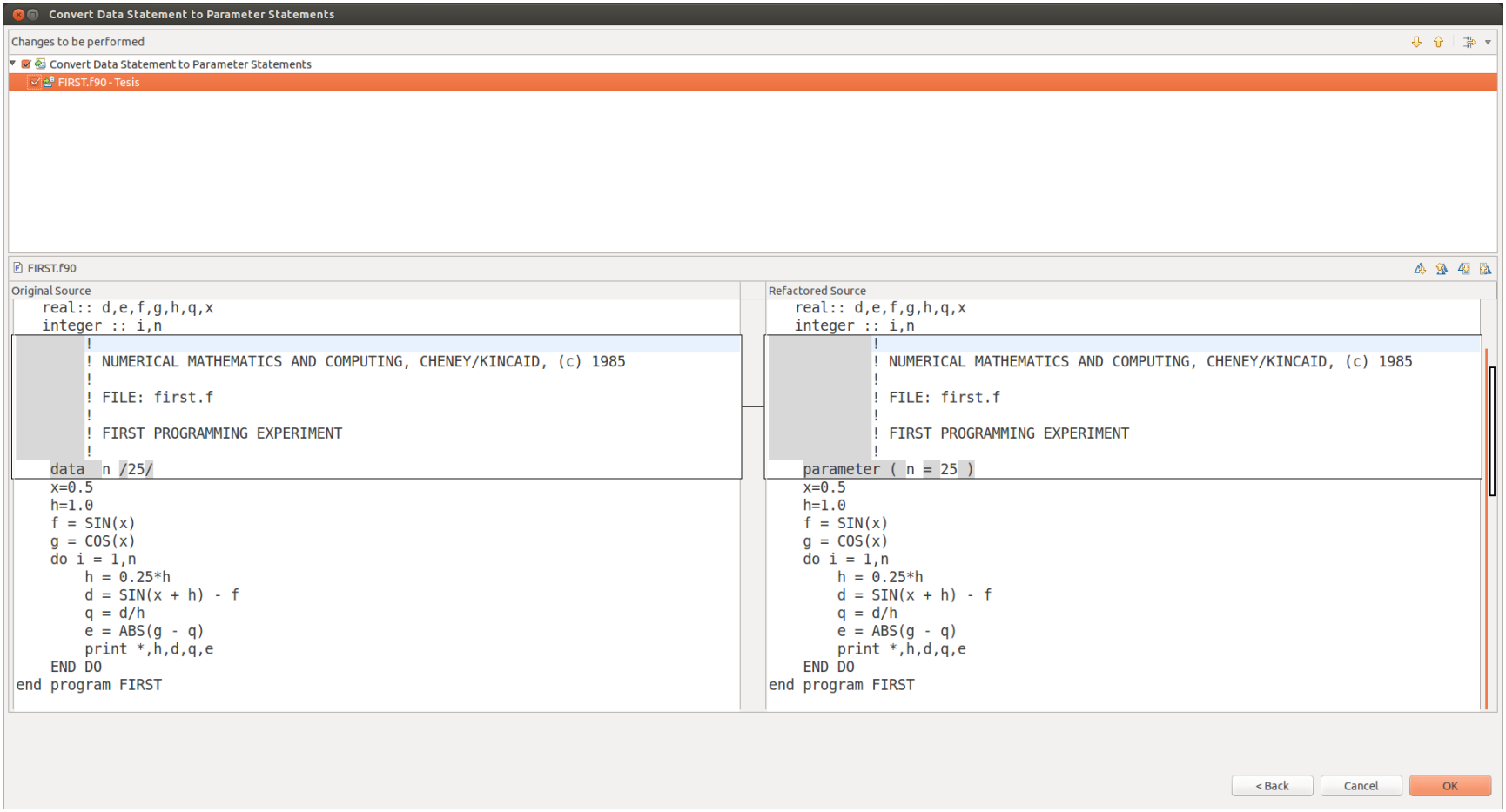


Figura 7.67: Vista Previa en Diferencia (DIFF VIEW) del Código Resultante de la Transformación de la Iteración 10

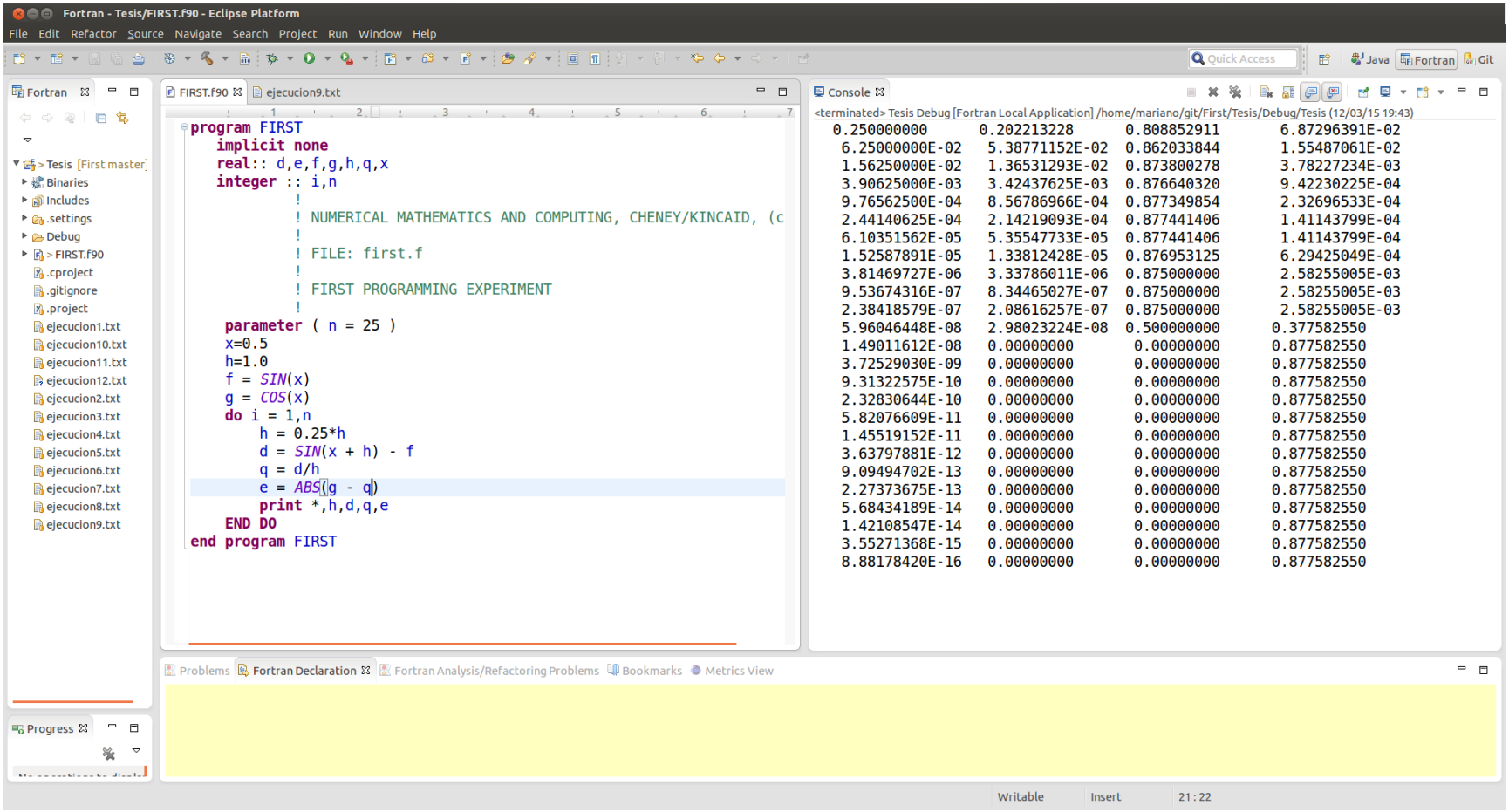


Figura 7.68: Resultados de la Corrida del Programa FIRST.f90 (Iteración 10)

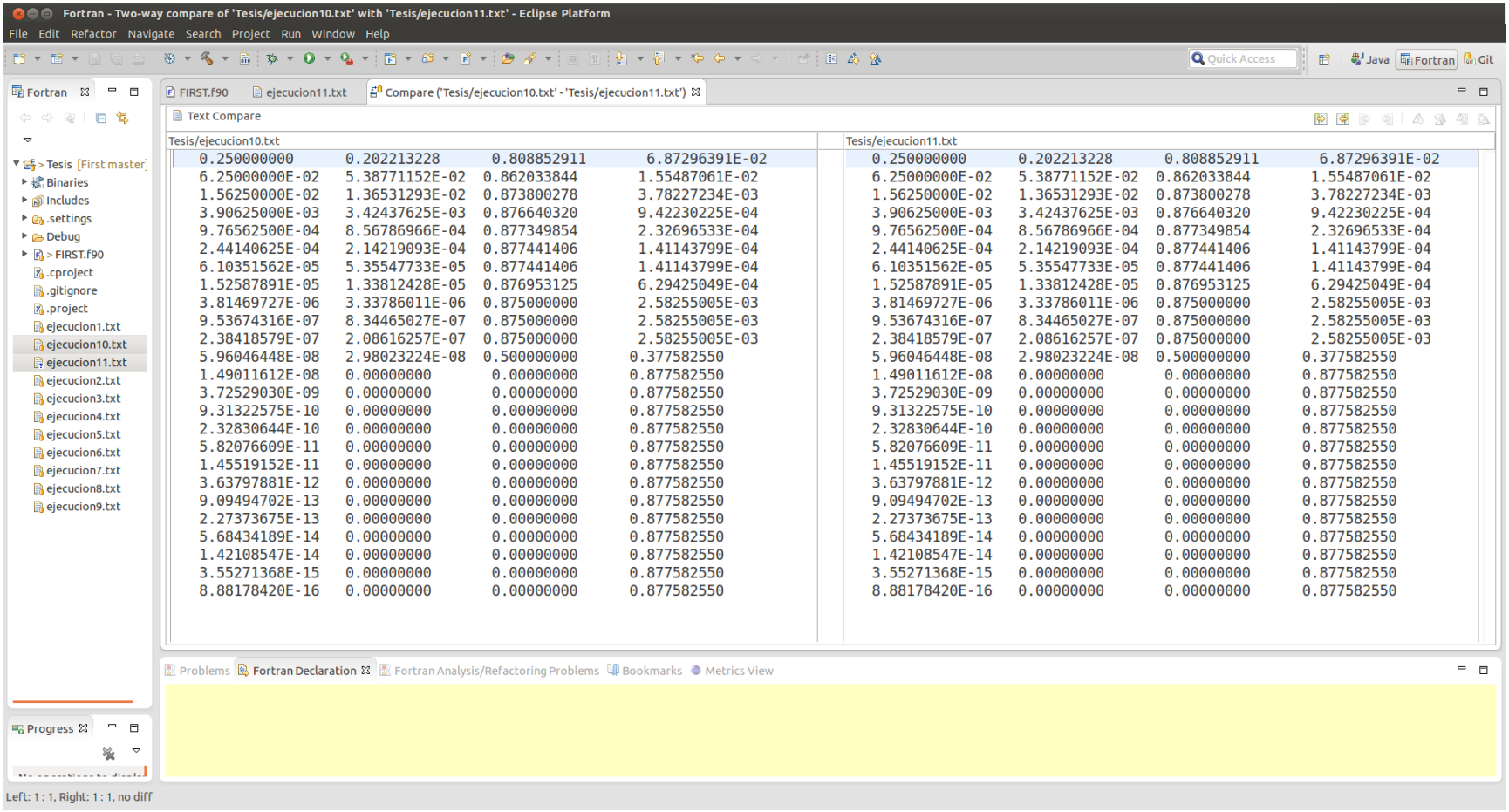


Figura 7.69: Vista de Diferencia (DIFF VIEW) de la Comparación de los Resultados de la Corrida del Código Resultante de la Iteración 9 con el de la Iteración 10

Iteración 11 Resultado de la Retroalimentación

Esta iteración surge debido a la introducción de código fuente escrito en mayúsculas durante el proceso, por parte de ciertas transformaciones. Por ello se debe volver a aplicar esta transformación para mantener la coherencia entre mayúsculas y minúsculas a lo largo del código fuente. ver Figura 7.70, Figura 7.72, Figura 7.71, Figura 7.74 y Figura 7.75

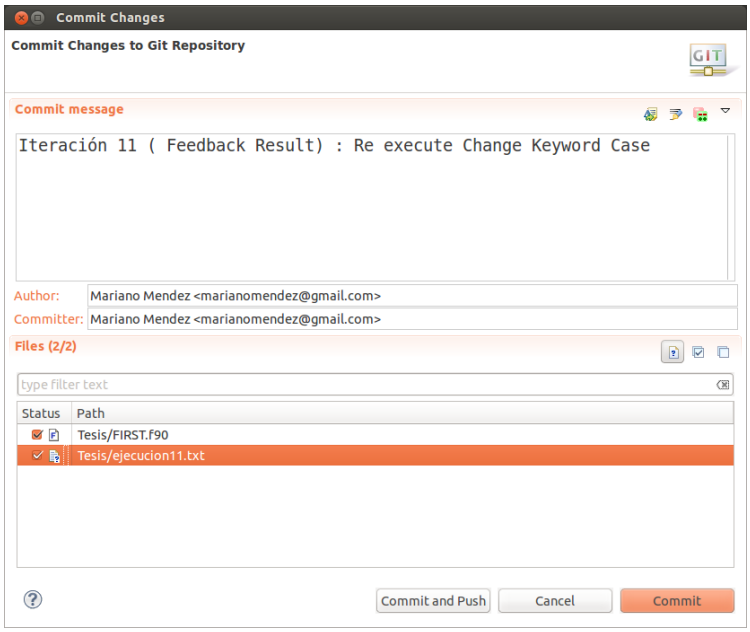


Figura 7.70: Commit de la Versión de Trabajo

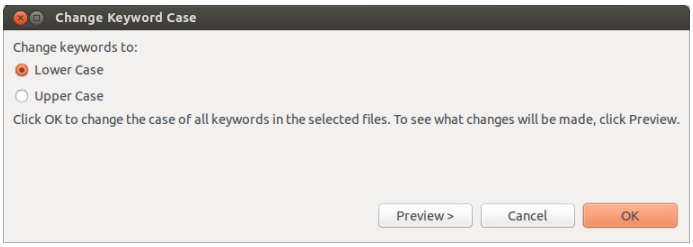


Figura 7.71: UI para Seleccionar el Tipo de Cambio

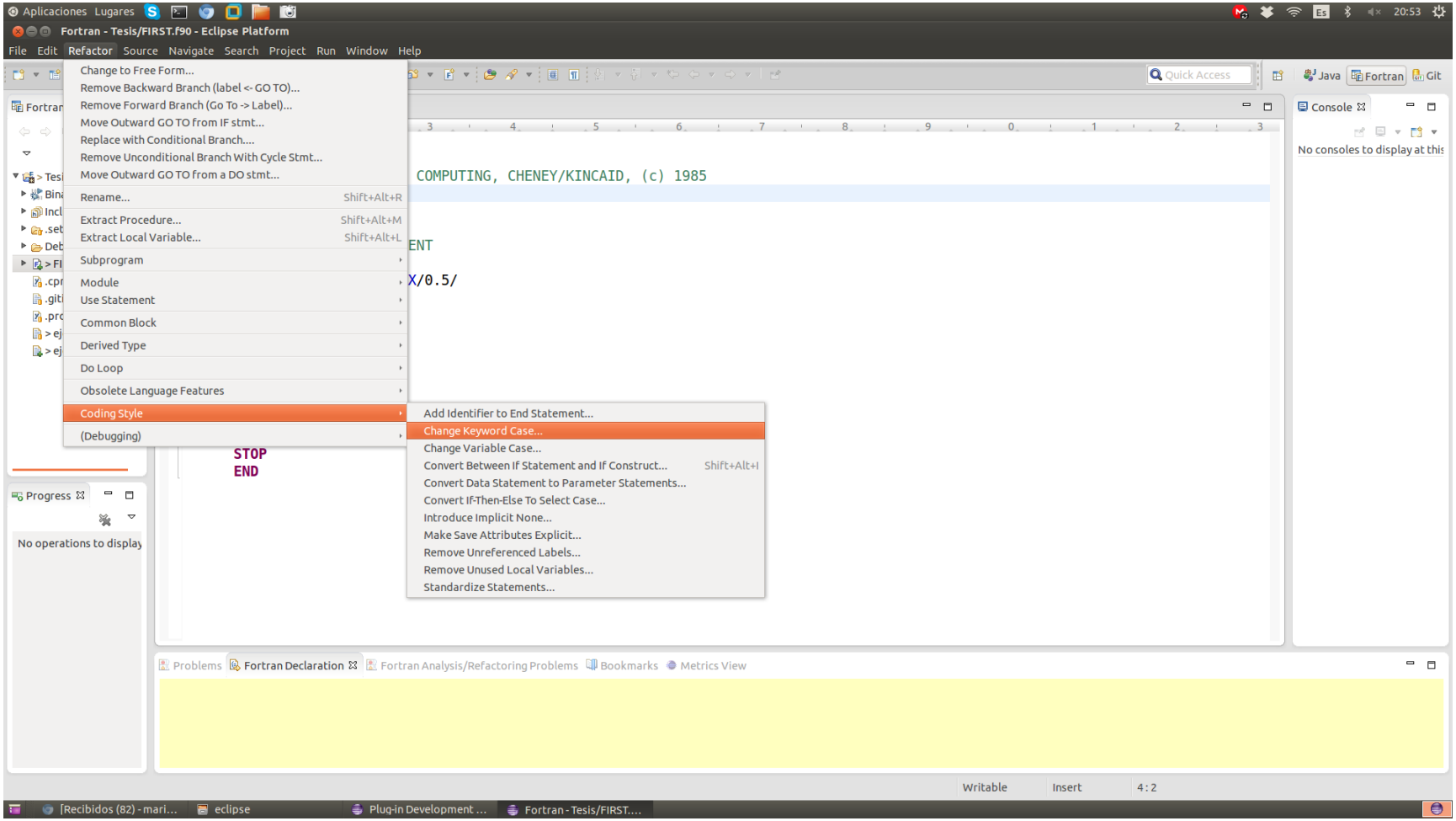


Figura 7.72: Opción de Menú de Transformaciones del IDE

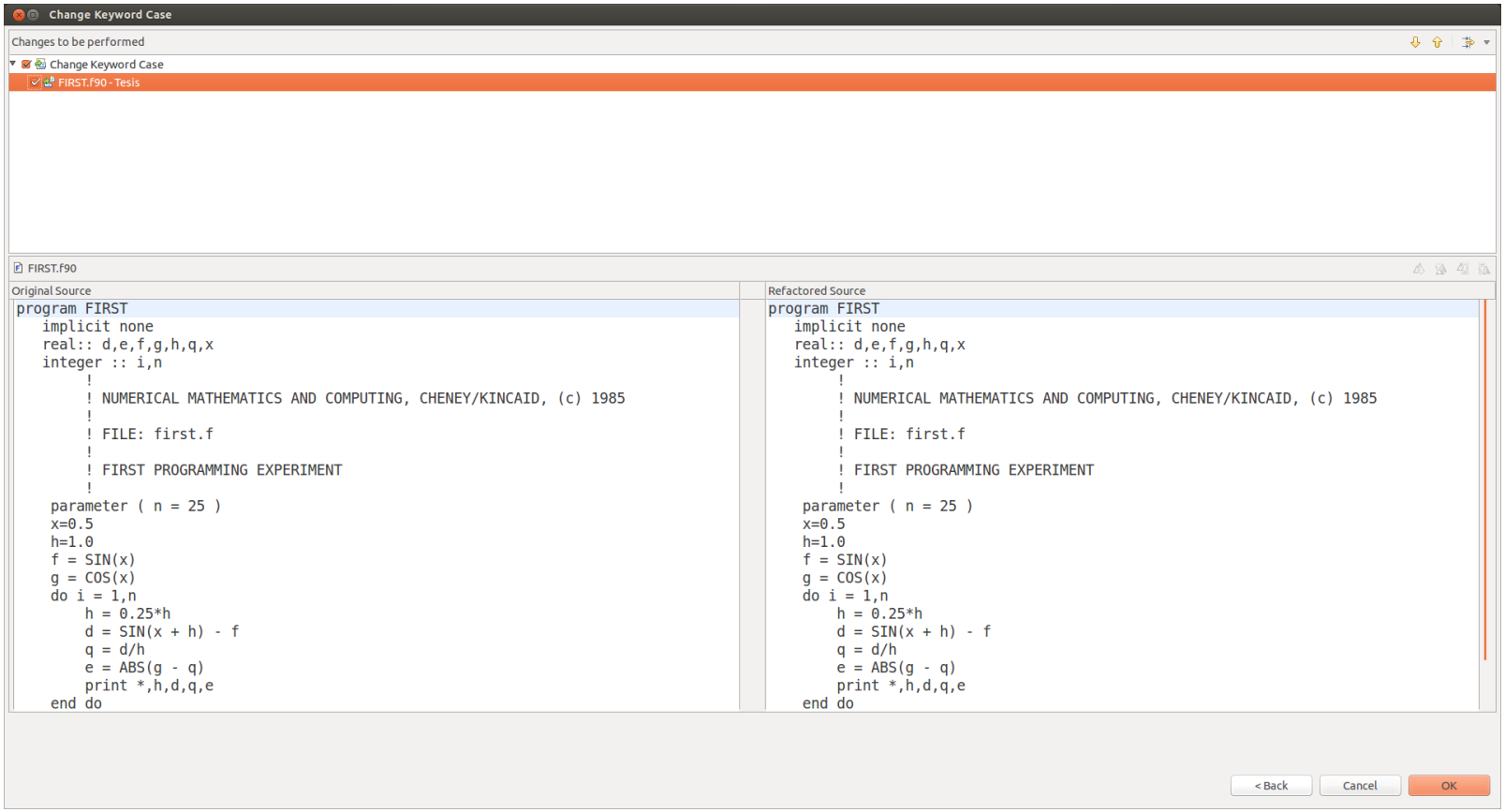


Figura 7.73: Vista de Diferencia (DIFF VIEW) en la que se Aprecia el Antes y el Después de la Aplicación de la Transformación de Código Fuente Iteración 11

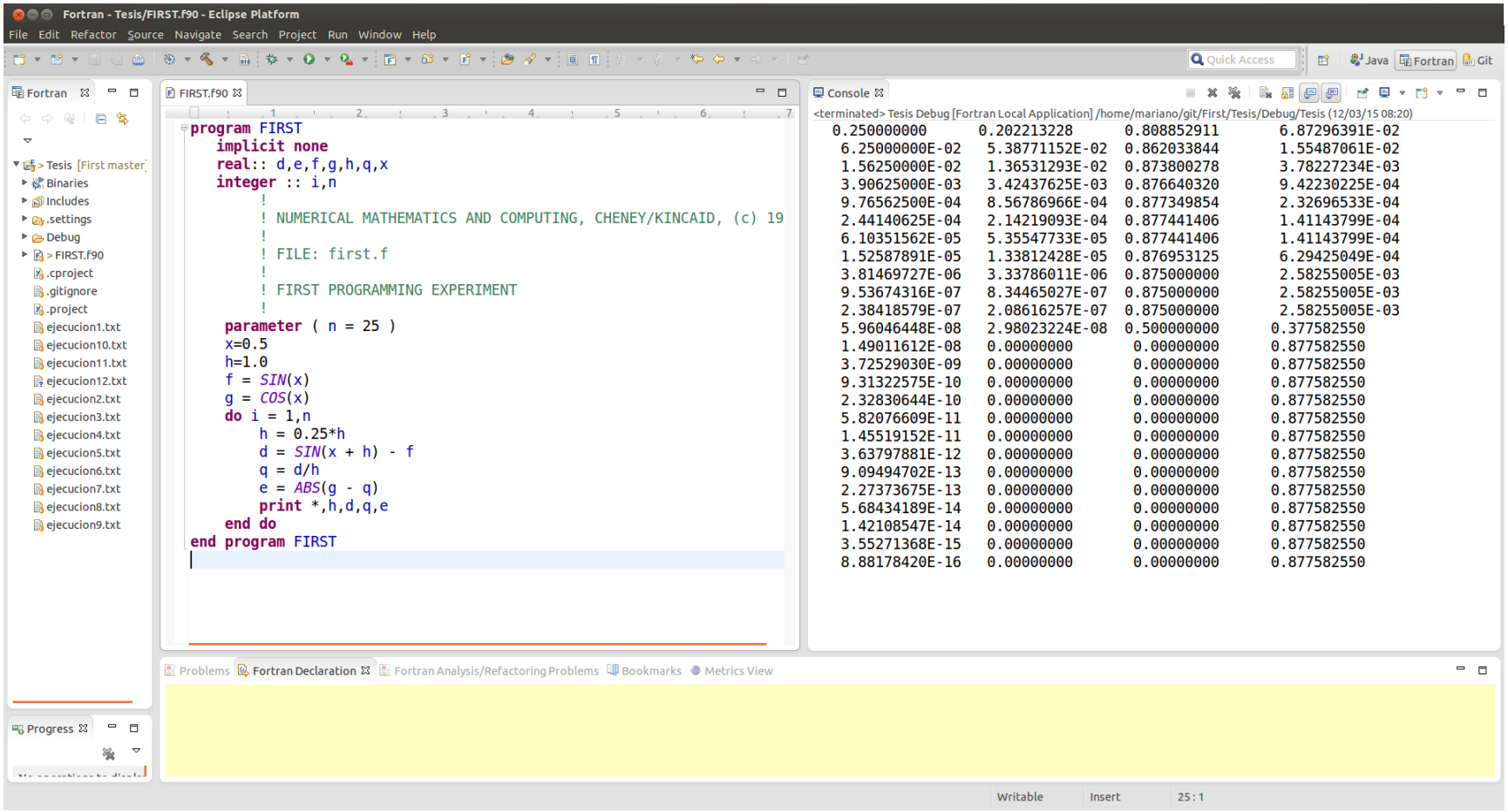


Figura 7.74: Resultados Obtenidos en la Corrida de FIRST.f90 en la Iteración

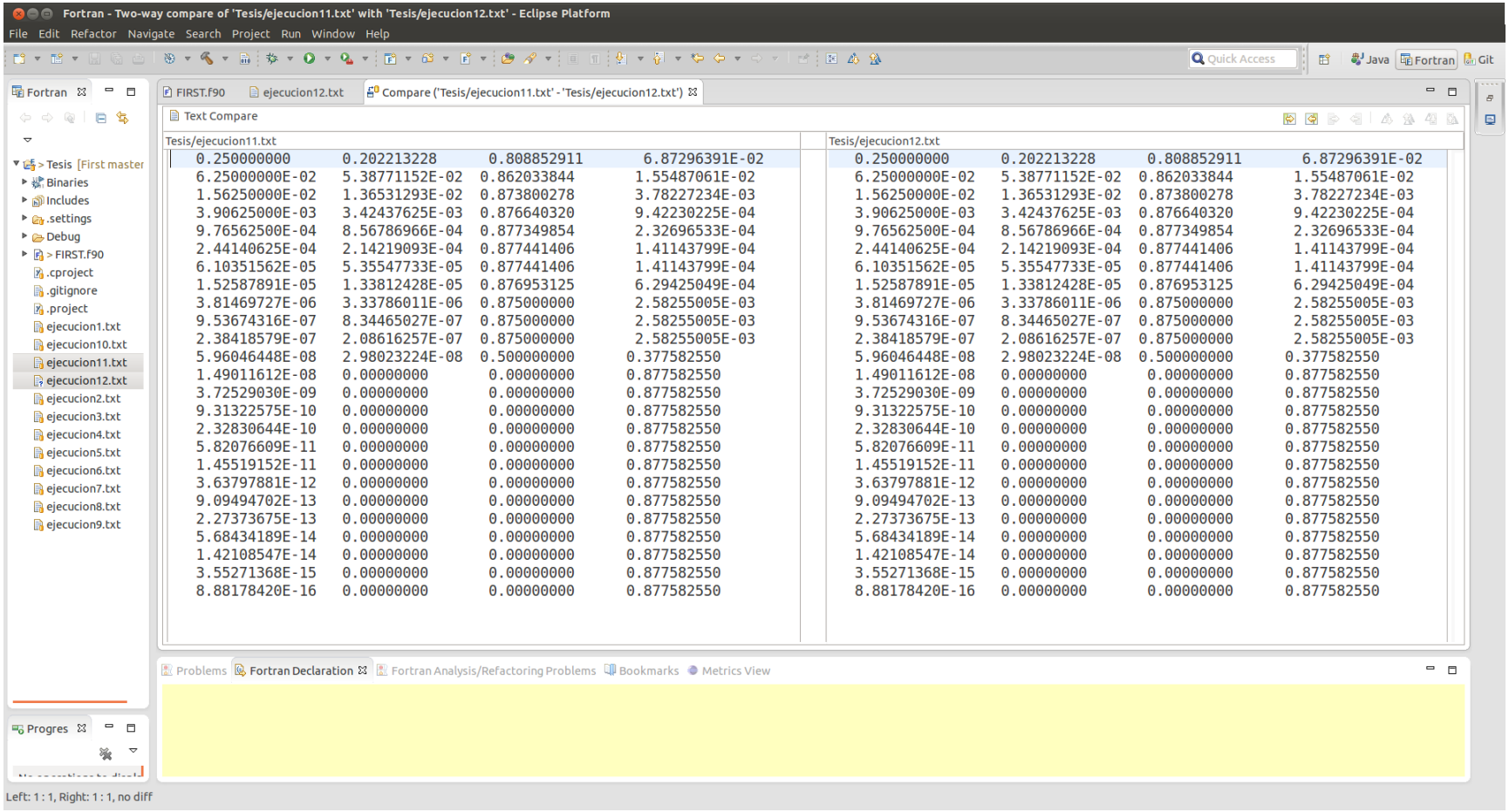


Figura 7.75: Se Comparan los Resultados de las Dos Versiones del Programa, la Original FIRST.f vs la Transformada FIRST.f90

Análisis de los Resultados Obtenidos en el Proceso

Partiendo de un programa escrito íntegramente en FORTRAN 77 (ver Figura 7.1):

```
      PROGRAM FIRST
C NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
C
C FILE: first.f
C
C FIRST PROGRAMMING EXPERIMENT
C
      DATA N/25/, H/1.0/, X/0.5/
      F = SIN(X)
      G = COS(X)
      DO 2 I = 1,N
      H = 0.25*H
      D = SIN(X + H) - F
      Q = D/H
      E = ABS(G - Q)
      PRINT *,H,D,Q,E
2     CONTINUE
      STOP
      END
```

Figura 7.76: Código Fuente del Programa FIRST.f90

se ha logrado llegar a una versión del mismo programa escrito en Fortran 90 (ver Figura 7.77). Este proceso ha sido realizado por 11 iteraciones en las que se han utilizado 9 transformaciones automáticas de código fuente implementadas como refactorizaciones integradas a un IDE, en este caso Eclipse.

```
program FIRST
     implicit none
     real:: d,e,f,g,h,q,x
     integer :: i,n
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
     parameter ( n = 25 )
     x=0.5
     h=1.0
     f = SIN(x)
     g = COS(x)
     do i = 1,n
          h = 0.25*h
          d = SIN(x + h) - f
          q = d/h
          e = ABS(g - q)
          print *,h,d,q,e
     end do
end program FIRST
```

Figura 7.77: Código Fuente del Programa FIRST.f90 Resultado de las 11 Iteraciones del Proceso de Desarrollo Dirigido por el Cambio (CDD)

Un aspecto notable se encuentra al revisar la quinta edición del mismo libro donde se extrajo el ejemplo con el que hemos trabajado, y en el cual los autores han modernizado a Fortran 90 los ejemplos. Esto puede apreciarse en la Figura 7.78. Haciendo una comparación entre el resultado de las 11 iteraciones (proceso automatizado) y el ejemplo propuesto en la quinta edición del libro (escrito manualmente) puede verse que son prácticamente iguales, a excepción de agregados que han introducido los autores como la línea 7 del programa que no estaba en la versión de FORTRAN77. En conclusión, a partir de un programa escrito 1985 hemos arribado a una versión similar mediante la aplicación de sucesivos cambios automatizados a la escrita en el 2003 por los mismos programadores. *Aspectos destacables del proceso:*

1. La mayor parte de las transformaciones utilizadas han sido aplicadas de forma automática, es decir, el programador no ha llevado a cabo dichas transformaciones. Esto hace que dadas las características de las transformaciones se garantice el cumplimiento de las precondiciones y por ende, queda asegurado que podrá transformarse solamente código que cumple dichas condiciones.

2. La fase de verificación en cada iteración permite asegurar que la funcionalidad del programa no ha cambiado y que las transformaciones no han aportado efectos secundarios o colaterales que se vean reflejados en los resultados numéricos.

3. La fase de Retroalimentación ha sido de ayuda para modificar la lista de cambios iniciales según el proceso lo requiera. Ha permitido además establecer la necesidad de nuevas transformaciones a ser implementadas para ser utilizadas en casos futuros.

Desde el punto de vista de la utilización del flujo de trabajo éste ha resultado ser de mucha ayuda actuando de hilo conductor del proceso de transformación. Queda claro que cuanto mayor sea la cantidad de transformaciones disponibles y cuanto mayor sea cantidad de herramientas de análisis de código fuente integradas, más interesante resultará la aplicación de las mismas, ya sea desde el punto de vista del Mantenimiento de Software como desde el punto de vista tradicional de la Construcción desde cero del mismo. Por otro lado se puede observar la complejidad que requiere este tipo de proceso, teniendo en cuenta que el programa analizado ocupa no más de unas 15 líneas de código fuente. Podría hacerse una extrapolación del grado de complejidad en programas donde su estructura está formada por cientos de miles de líneas de código fuente o hasta incluso millones de líneas de código. A continuación se lista la versión del mismo programa que aparece en la quinta edición del libro [47], ver Figura 7.78.

```
! Numerical Mathematics and Computing, Fifth Edition
! Ward Cheney & David Kincaid
! Brooks/Cole Publ. Co.
! Copyright (c) 2003.  All rights reserved.
! For educational use with the Cheney–Kincaid textbook.
! Absolutely no warranty implied or expressed.
!
! Section 1.1
!
! File: first.f90
!
! First programming experiment:
compute derivative of sin(x)
! by limit definition

program first
        integer, parameter :: n =25
        integer :: i
        real :: error, h, y

        x = 0.5
        h = 1.0
        print*, "  i    h    y    error"
        do i = 1,n
                h = 0.25*h
                y = (sin(x + h) - sin(x))/h
                error = abs(cos(x) - y)
                print *, i, h, y, error
        end do
end program first
```

Figura 7.78: Código Fuente del Programa Editado en la Quinta Edición del Libro de Cheney-Kincaid

7.2. Resumen

En este capítulo se ha aplicado el proceso de desarrollo dirigido por el cambio como así también el flujo de trabajo propuesto en cada una de las once iteraciones en la se ha actualizado un programa escrito inicialmente en FORTRAN 77 a una versión más moderna del lenguaje como es Fortran 90. Estas iteraciones han sido realizadas en el ejemplo de la Sección 7.1.

Capítulo 8

Aplicación del Proceso: Caso de Estudio 2

En este capítulo se estudiará una segunda aplicación del proceso propuesto paso a paso con un ejemplo de una aplicación científica real. En cada paso se mostrarán las fases del proceso y el flujo de trabajo propuestos en los capítulos anteriores.

8.1. Caso de Estudio 2: Introducción

Si bien el caso de estudio anterior pudiera parecer sencillo, cuando abordamos una caso de estudio de un programa que es utilizado en la vida real, la complejidad puede incrementarse exponencialmente. En este caso se ha elegido un pequeño programa publicado en un trabajo del 2010 [64] referido a la solución encontrada por Kerr que describe el espacio-tiempo en la ergosfera o vecindad de los agujeros negros en rotación, a los que ahora se llama "agujeros negros de Kerr" y a la solución de la ecuación se la conoce como "métrica de Kerr" o "solución de Kerr". La descripción de los agujeros negros en rotación representa una contribución destacada a la astrofísica, ya que se piensa que la mayoría de los agujeros negros están animados por un movimiento de rotación suficientemente importante para

que éste tenga una influencia directa sobre su medio ambiente inmediato [123]. Este programa está escrito íntegramente en Fortran y consta de unas 3600 líneas de código fuente y puede ser obtenido en http://www.astro.washington.edu/users/agol/geokerr/.

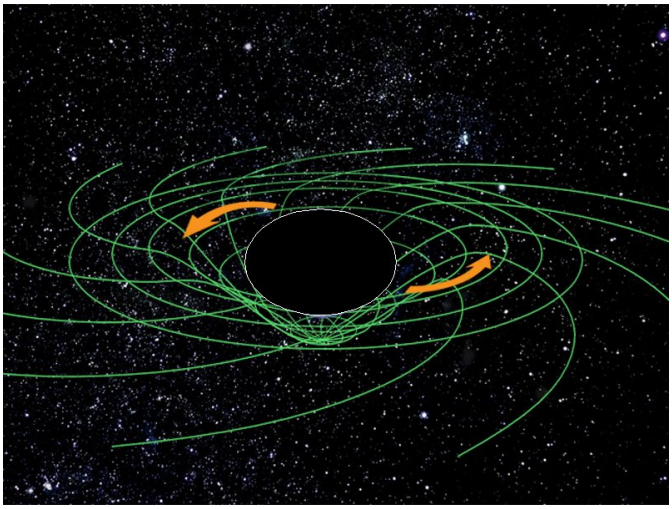


Figura 8.1: Agujero Negro de Kerr Caracterizado por la Masa y el Espin, Extraido de http://www.if.ufrgs.br/~thaisa/bn/01_definicao.htm

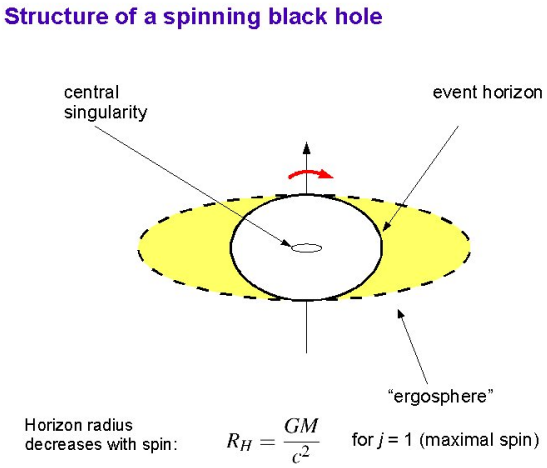


Figura 8.2: Descripción de un Agujero Negro de Kerr con su Singularidad, Extraido de http://www.if.ufrgs.br/~thaisa/bn/01_definicao.htm

En la Figura 8.4 se ve el grafo estático de llamada a funciones y subrutinas del programa y puede apreciarse su complejidad. Para determinar una lista inicial

de cambios a realizar se ha obtenido el reporte de características obsoletas del lenguaje que puede apreciarse en la Figura 8.6. Además se ha utilizado la *metric View* para determinar aquellas rutinas que por ser más complejas necesitan ser analizadas más en detalle, ver Figura 8.5. Debido a la complejidad que conlleva realizar este tipo de proceso se ha seleccionado una serie de cambios acotada, dejando de lado otros que por su dimensión no pueden ser descritos en este trabajo. Después de haber realizado un análisis del código fuente se determinó una lista inicial de cambios a realizar:

1. Cambiar a formato libre

2. Pasar las instrucciones a minúsculas.

3. Pasar las variables a minúsculas.

4. Eliminar las instrucciones DO escritas en formato antiguo.

Cabe destacar el hecho de que las herramientas de análisis utilizadas permiten validar sus propios resultados pues de la Figura 8.5 se desprende el análisis de la vista de métricas -donde se muestran la complejidad ciclomática e índice de mantenibilidad- que las rutinas más complejas son: TESTGEOPHIT(), GEOKERR(), geomu() y geor(). Las primeras dos rutinas mencionadas poseen un gran número de instrucciones DO escritas en el formato antiguo.

Siguiendo el proceso y el flujo de trabajo planteado se han definido 4 iteraciones iniciales: cambiar a formato libre, pasar las instrucciones a minúsculas, pasar las variables a minúsculas y eliminar las instrucciones DO escritas en formato antiguo.

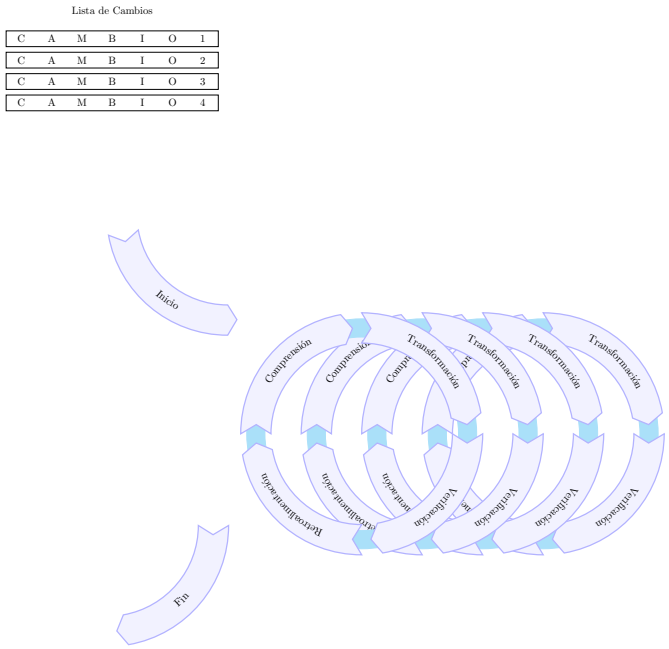


Figura 8.3: Proceso de Desarrollo Dirigido por el Cambio para geoekerr.F

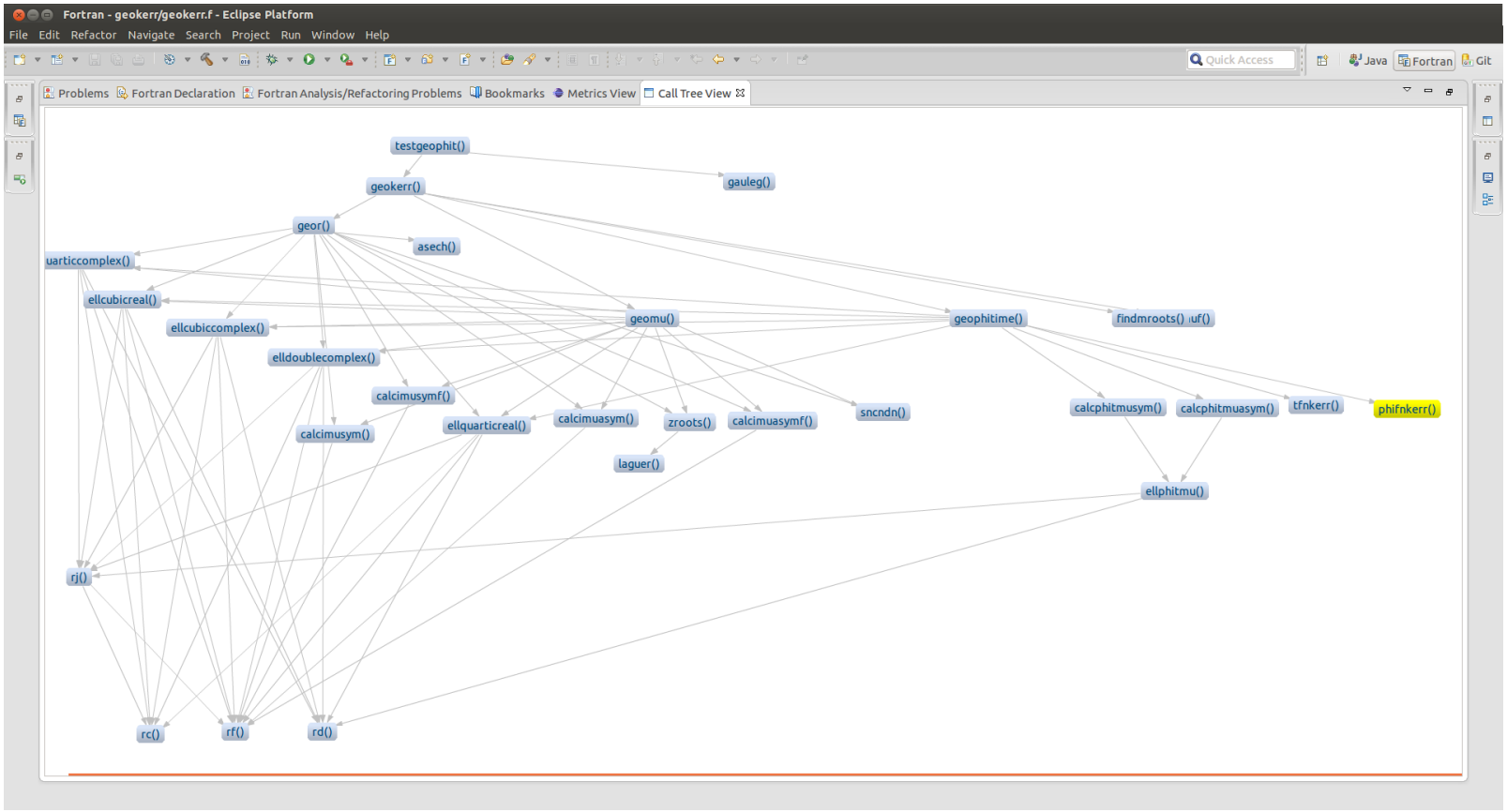


Figura 8.4: Grafo de Llamadas Estático del Programa geokerr.f, Herramienta Implementada e Integrada a Eclipse en este Trabajo

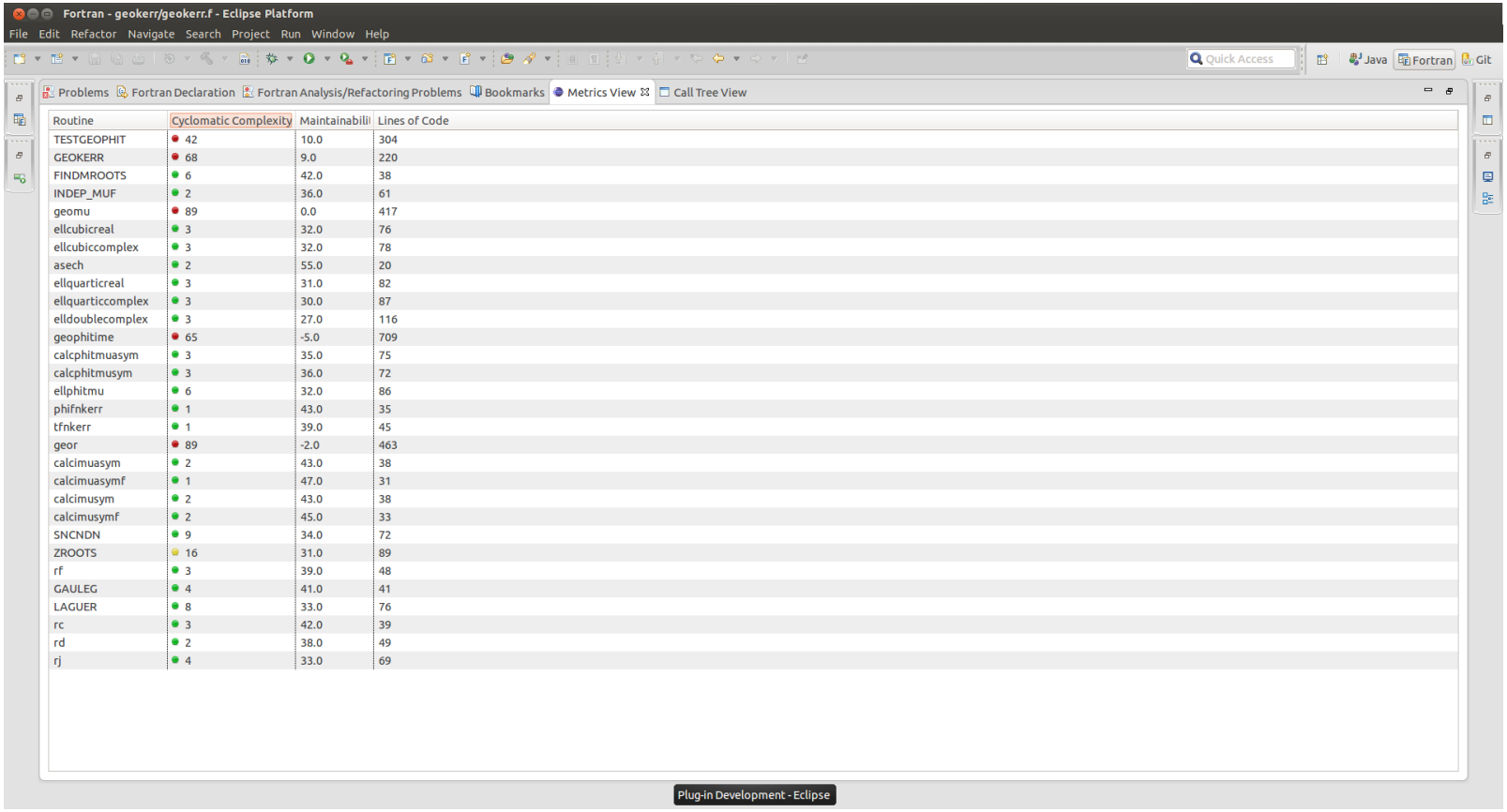


Figura 8.5: Resultado del Análisis del Código Fuente del Programa geokerr.f, Herramienta Implementada e Integrada a Eclipse en este Trabajo

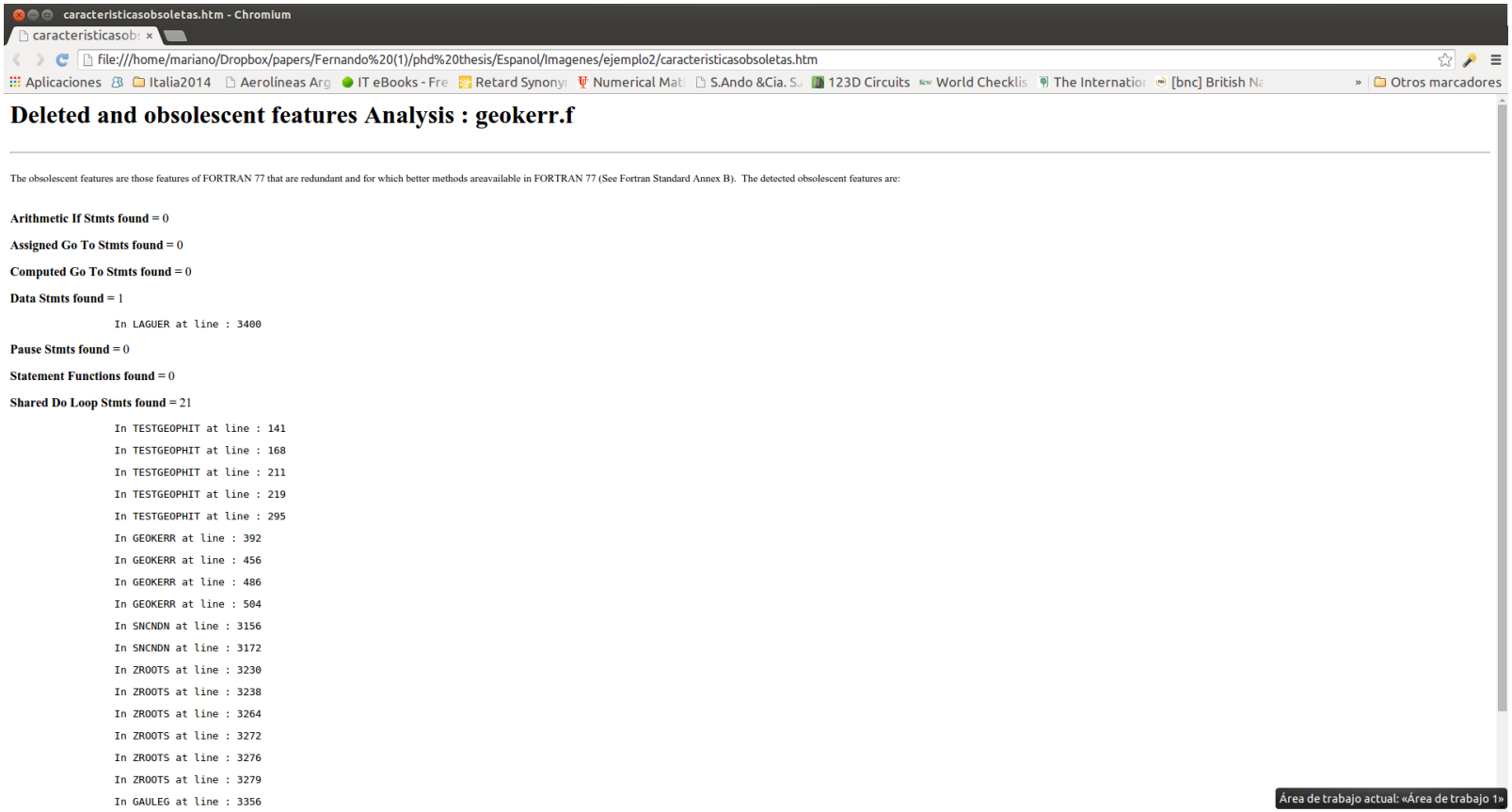


Figura 8.6: Listado de Características Obsoletas del Lenguaje del Programa geokerr.f, Herramienta Implementada e Integrada a Eclipse en este Trabajo

Luego de llevar a cabo las cuatro iteraciones propuestas se ha agregado una última iteración. Tras el análisis de retroalimentación de cada una de las iteraciones, se ha incluido una nueva iteración en la cual se intentara mejorar el rendimiento de la aplicación. En esta iteración se planteará la utilización de una transformación automática de código fuente cuyo objetivo es paralelizar una o varias instrucciones DO de las rutinas que más computo requieran, con el objetivo de mejorar el rendimiento. Dado que existen varias formas para intentar mejorar el rendimiento del software desde la aplicación de paralelización de memoria compartida, paralelización utilizando Message Passing Interface, mejoras del rendimiento aplicando optimizaciones al código fuente (optimización en cantidad de operaciones a utilizar, inlinning functions, etc.), el método elegido se basa en la aplicación de paralelismo de memoria compartida utilizando la biblioteca OpenMp [4]. El cambio se llevará a cabo en la iteración 5, y será descrito en detalle.

Iteración 1: Cambiar a formato libre

Al igual que en el ejemplo anterior el primer cambio a realizar es pasar de formato fijo de Fortran a formato libre, teniendo en cuenta las desventajas que conlleva el formato fijo como por ejemplo que el formato fijo, que permite espacios ya sea en medio de una instrucción o del nombre de una variable. Para llevar a cabo la primera iteración se fija una versión inicial, ver Figura 8.8 utilizando Git como gestor de versiones. A continuación se ejecuta una corrida inicial para tener un patrón de validación de los resultados numéricos, ver Figura 8.7. Para ello se ha utilizado el siguiente archivo de entrada de datos utilizado en el artículo original [64] :

```
1
0.0D0
0.998d0
1.d2
2
-4.D0 8.D0 -6.D0 6.D0
```

```
10 10 1
STANDARD
MUO(1)
A
RCUT
NROTYPE
A1,A2,B1,B2
NRO,NPHI,NUP
```

Una vez obtenidos los datos se adjuntan al control de versiones y se procede a realizar la primera transformación de código fuente, ver Figura 8.8.

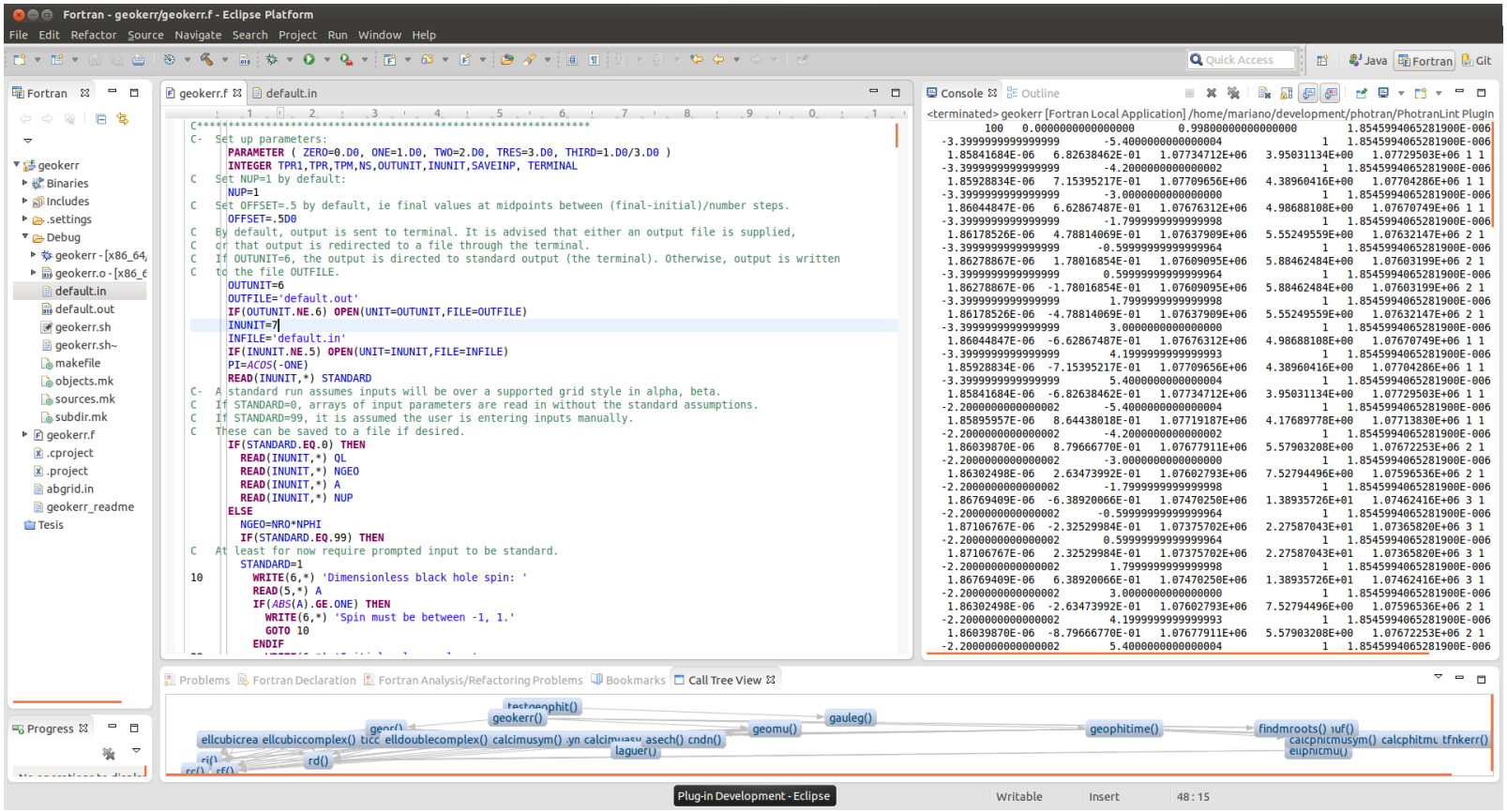


Figura 8.7: Listado de Características Obsoletas del Lenguaje del Programa geokerr.f

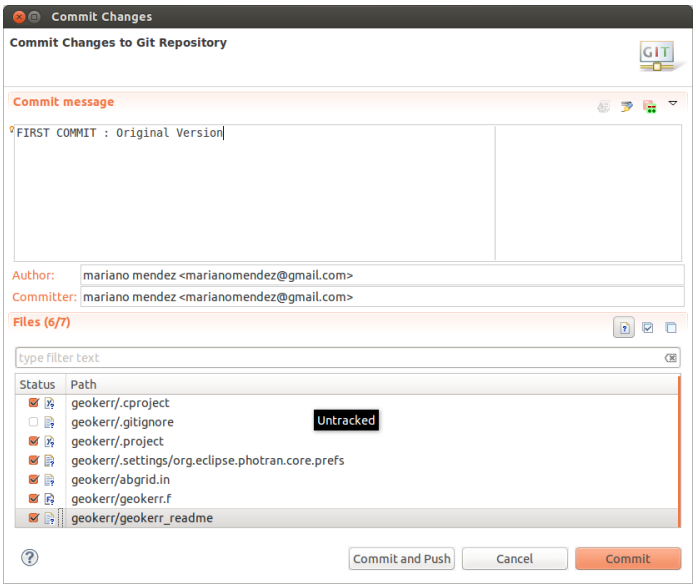


Figura 8.8: Commit de la Primera Versión del Programa al Iniciar el Ciclo de Iteraciones

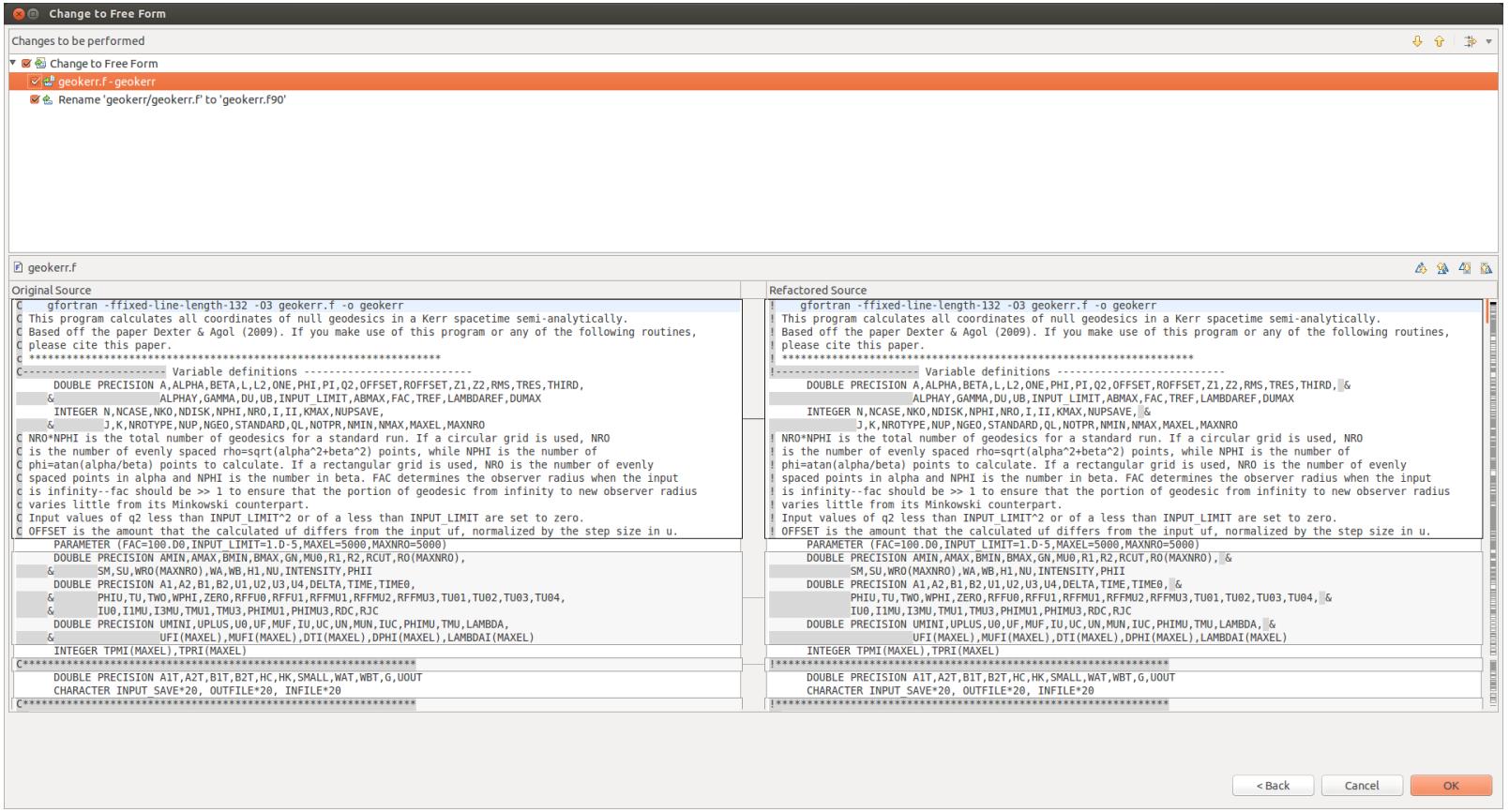


Figura 8.9: Vista de Diferencia (DIFF VIEW) en la que se Aprecia el Antes y el Después de la Aplicación de la Transformación de Código Fuente de la Iteración 1

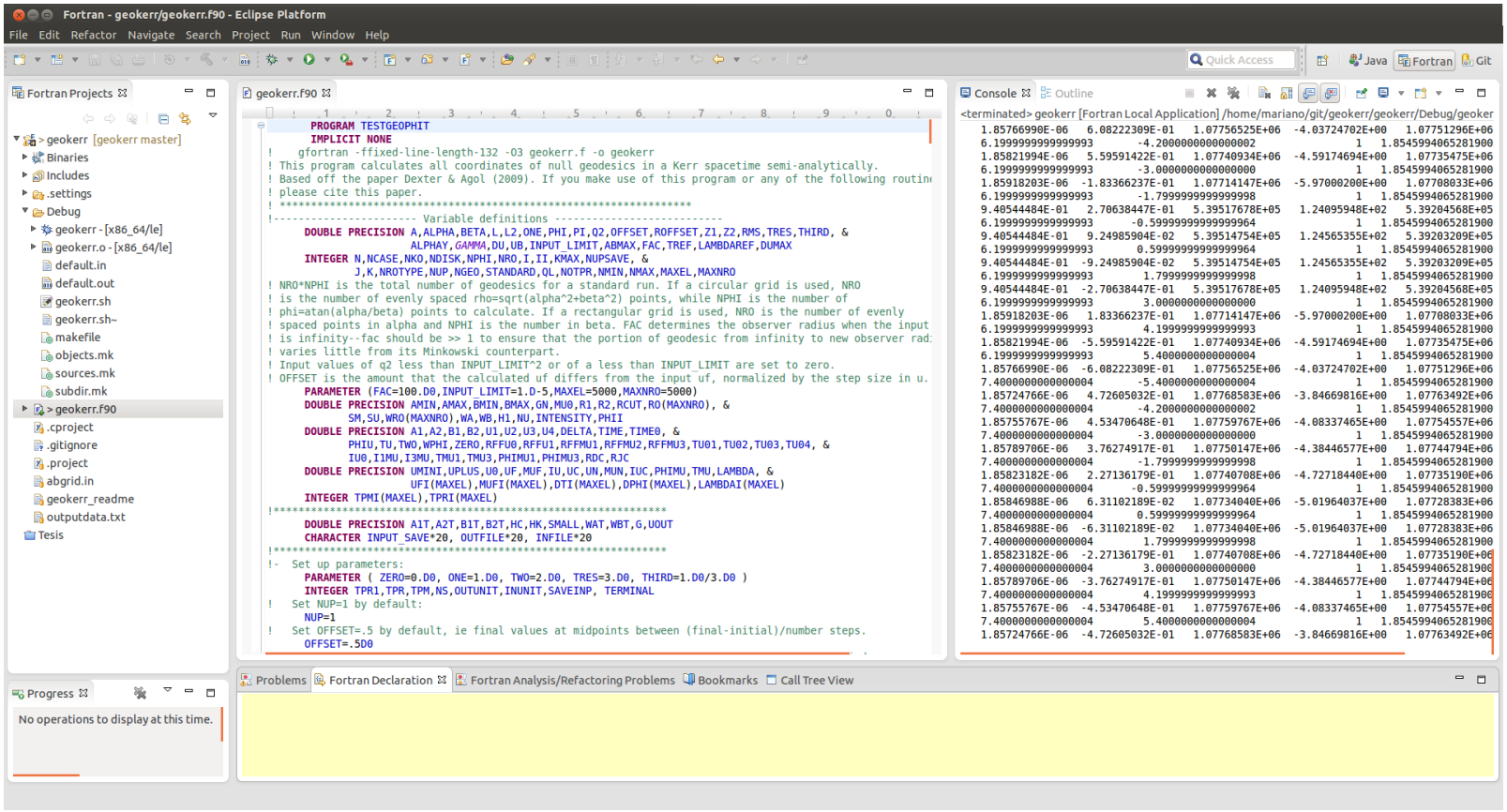


Figura 8.10: Resultados Obtenidos en la Corrida de geokerr.f90 en la Primera Iteración

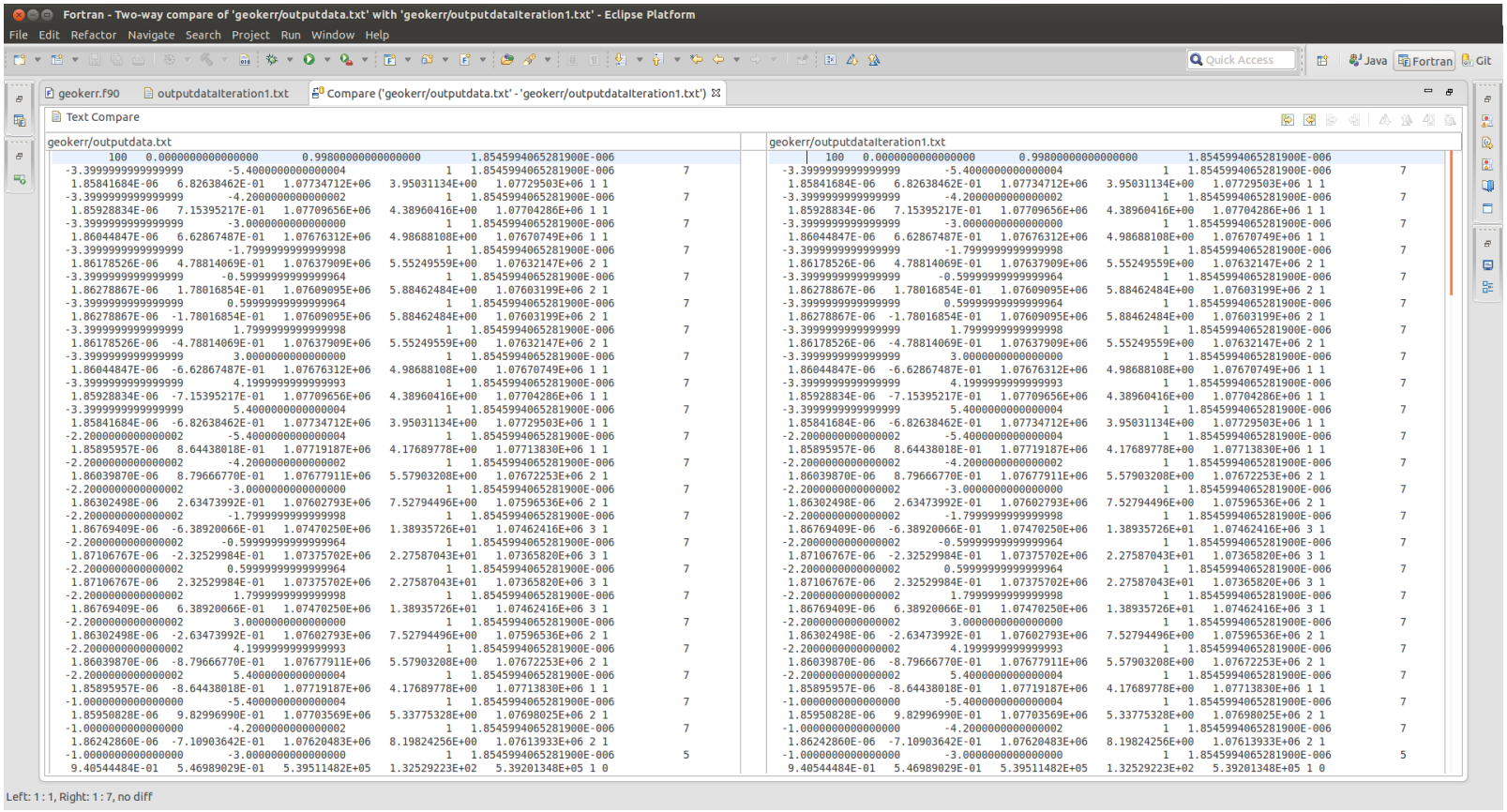


Figura 8.11: Se Comparan los Resultados de las Dos Versiones del Programa, la Original geokerr.f vs la Transformada geokerr.f90

Iteración 2: Pasar las instrucciones a minúsculas

En esta iteración se hará homogénea la escritura de las instrucciones del programa, ya que algunas partes del mismo está escrito en mayúsculas y otras en minúsculas. Para ello se utilizará la refactorización "Change Keyword Casez se comienza con el flujo de trabajo fijando la versión inicial de la iteración, ver Figura 8.12, Figura 8.13 Figura 8.14 y Figura 8.15.

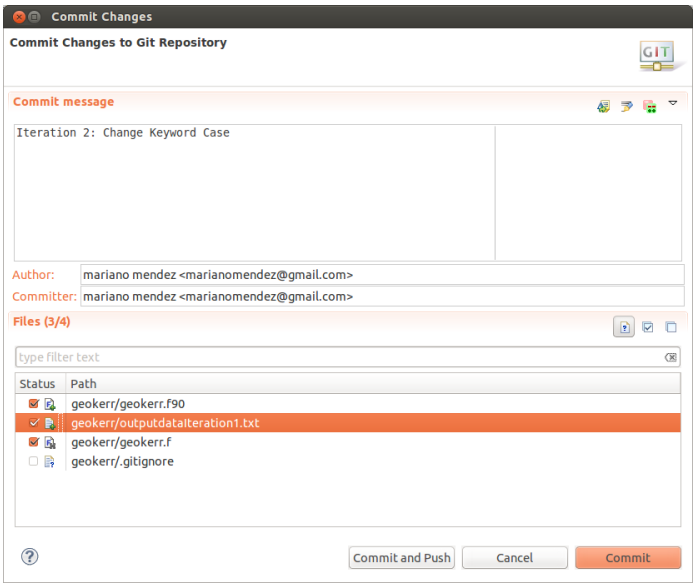


Figura 8.12: Commit de la Versión de la Iteración 1 del Programa al Iniciar la Iteración 2

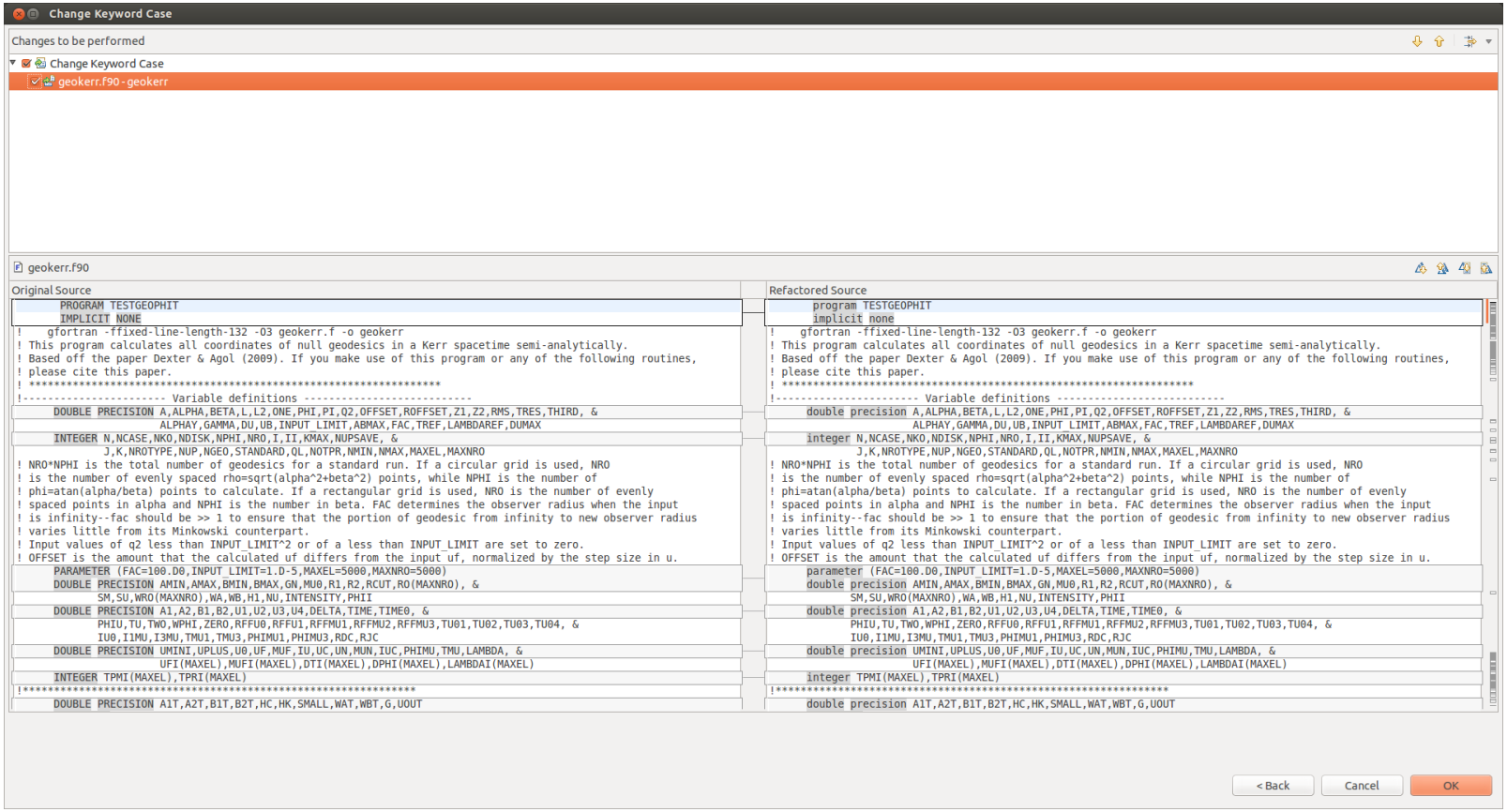


Figura 8.13: Vista de Diferencia (DIFF VIEW) en la que se Aprecia el Antes y el Después de la Aplicación de la Transformación de Código Fuente de la Iteración 2

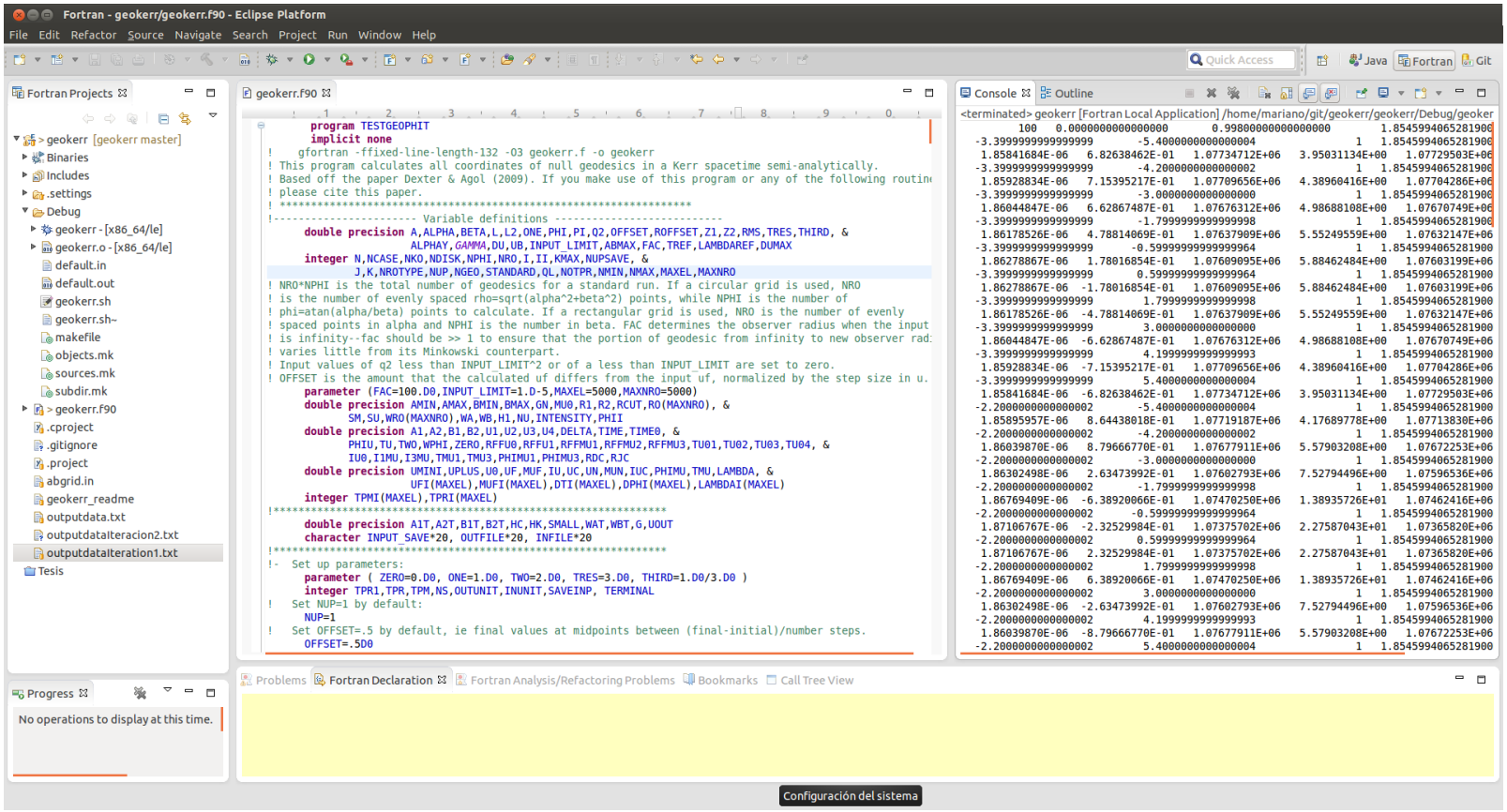


Figura 8.14: Resultados Obtenidos en la Corrida de geokerr.f90 en la Segunda Iteración

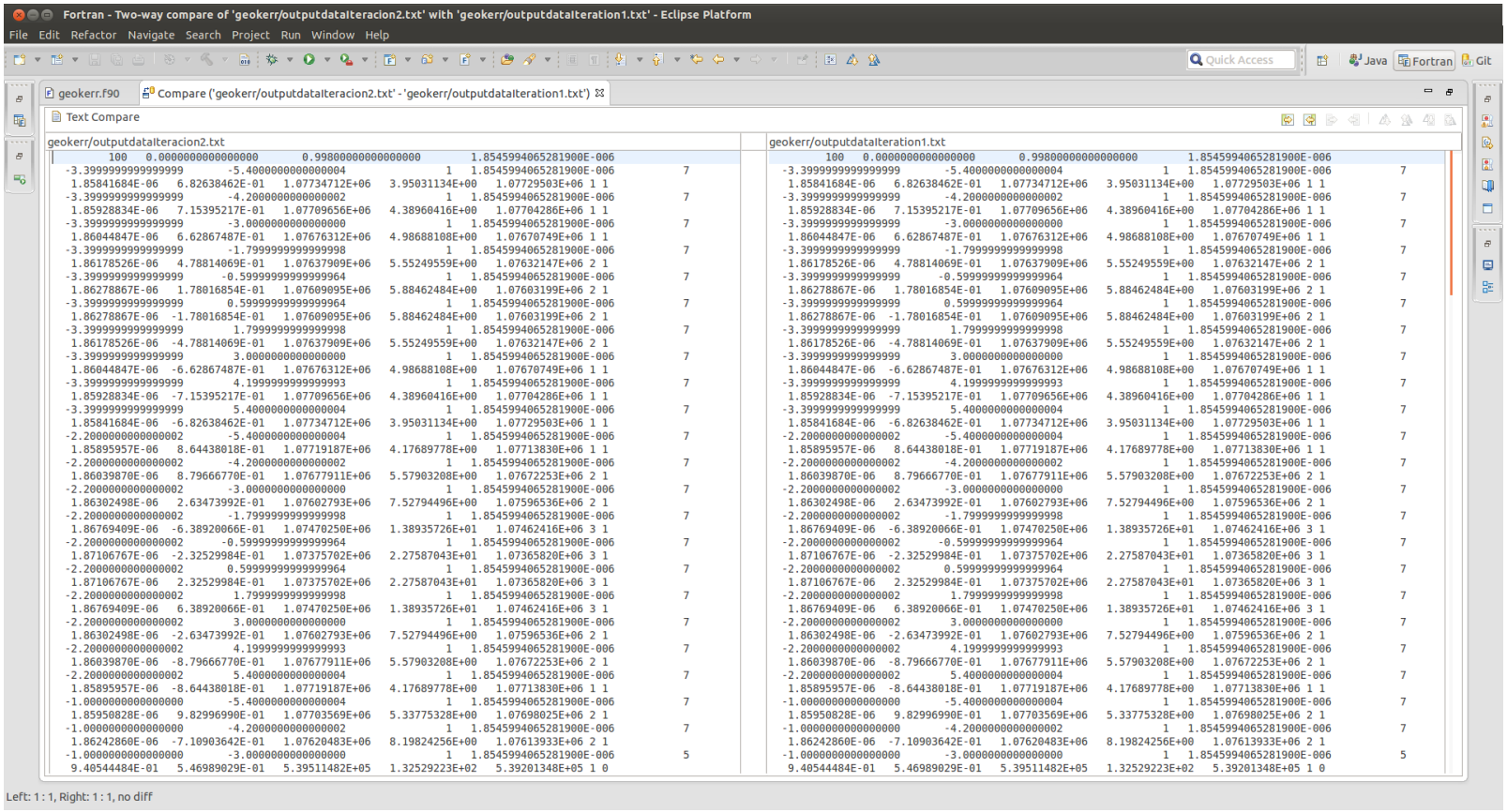


Figura 8.15: Se Comparan los Resultados de las Dos Versiones del Programa Transformado geokerr.f90 Entre la Iteración 1 y la 2

Iteración 3: Pasar las variables a minúsculas

En esta iteración se hará homogénea la escritura de las variables del programa, ya que algunas partes del mismo está escrito en mayúsculas y otras en minúsculas. Para ello se utilizará la refactorización "Change Variable Casez se comienza con el flujo de trabajo fijando la versión inicial de la iteración, ver Figura 8.16, Figura 8.17 Figura 8.18 y Figura 8.19.

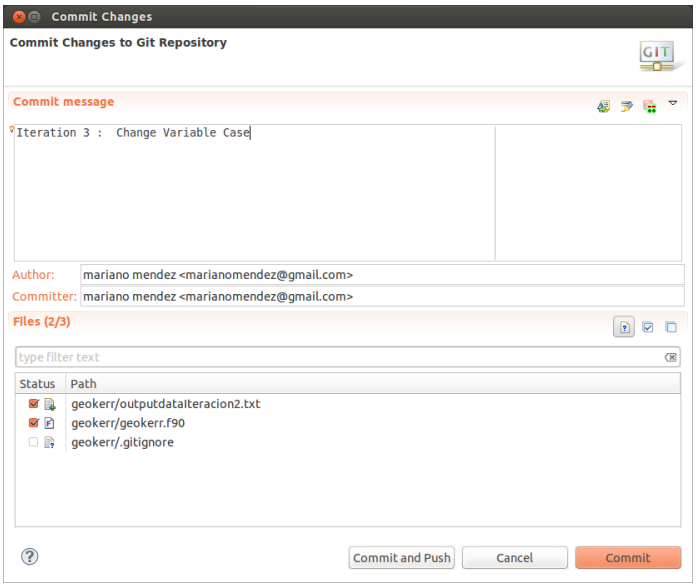


Figura 8.16: Commit de la Versión de la Iteración 2 del Programa al Iniciar la Iteración 3

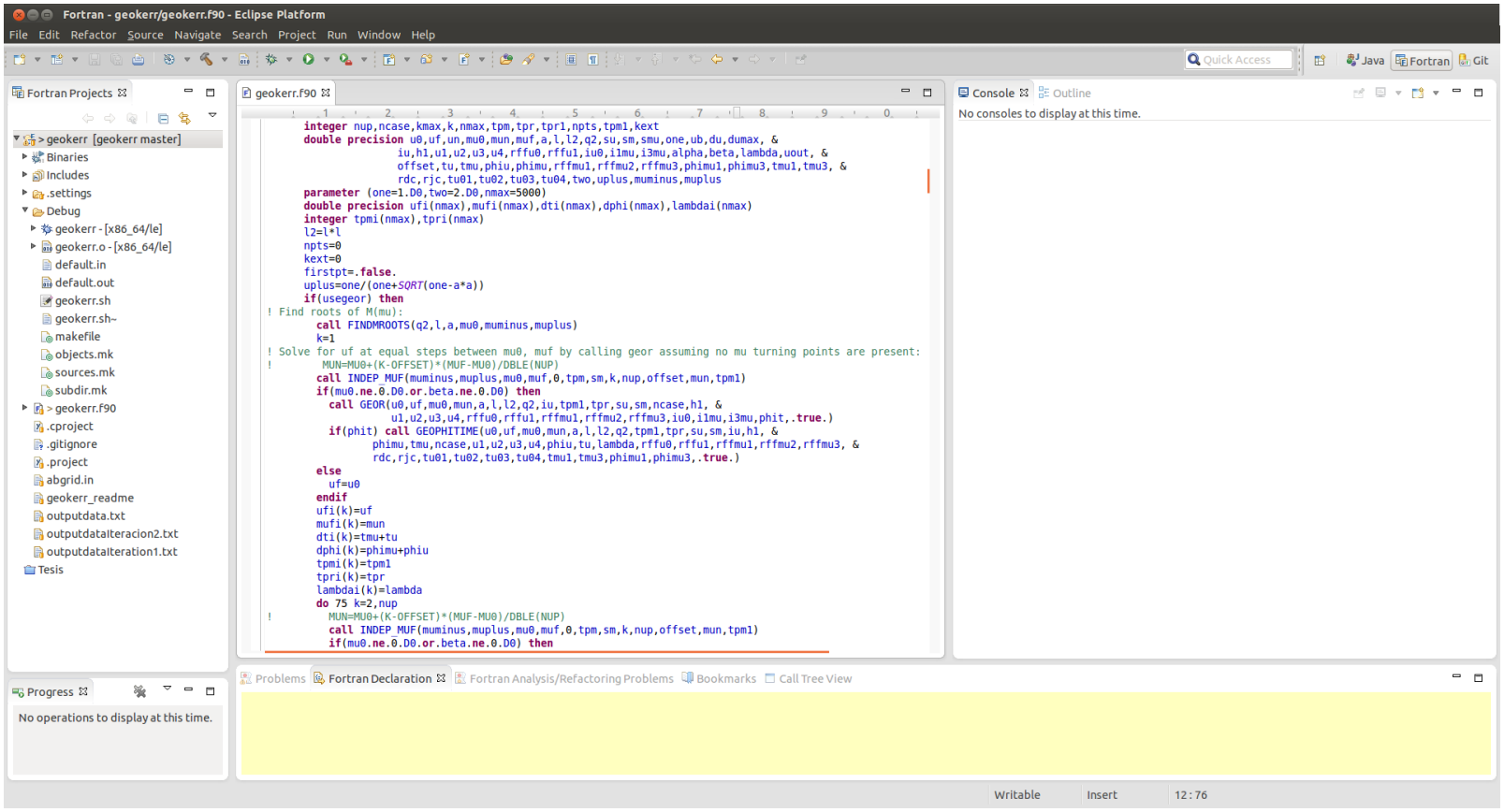


Figura 8.17: Editor del IDE Después de la Aplicación de la Transformación de Código Fuente de la Iteración 3

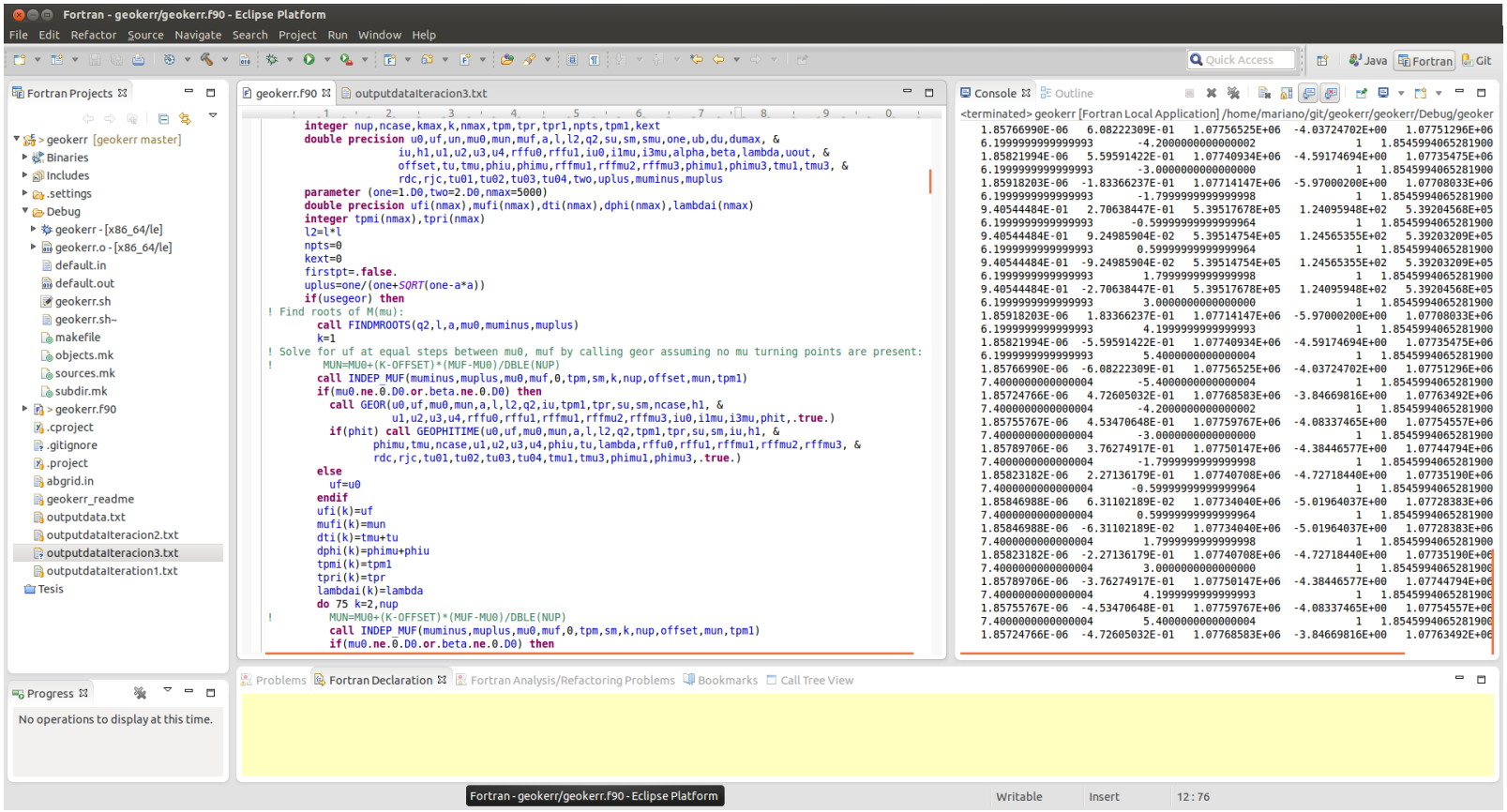


Figura 8.18: Resultados Obtenidos en la Corrida de geokerr.f90 en la Tercera Iteración

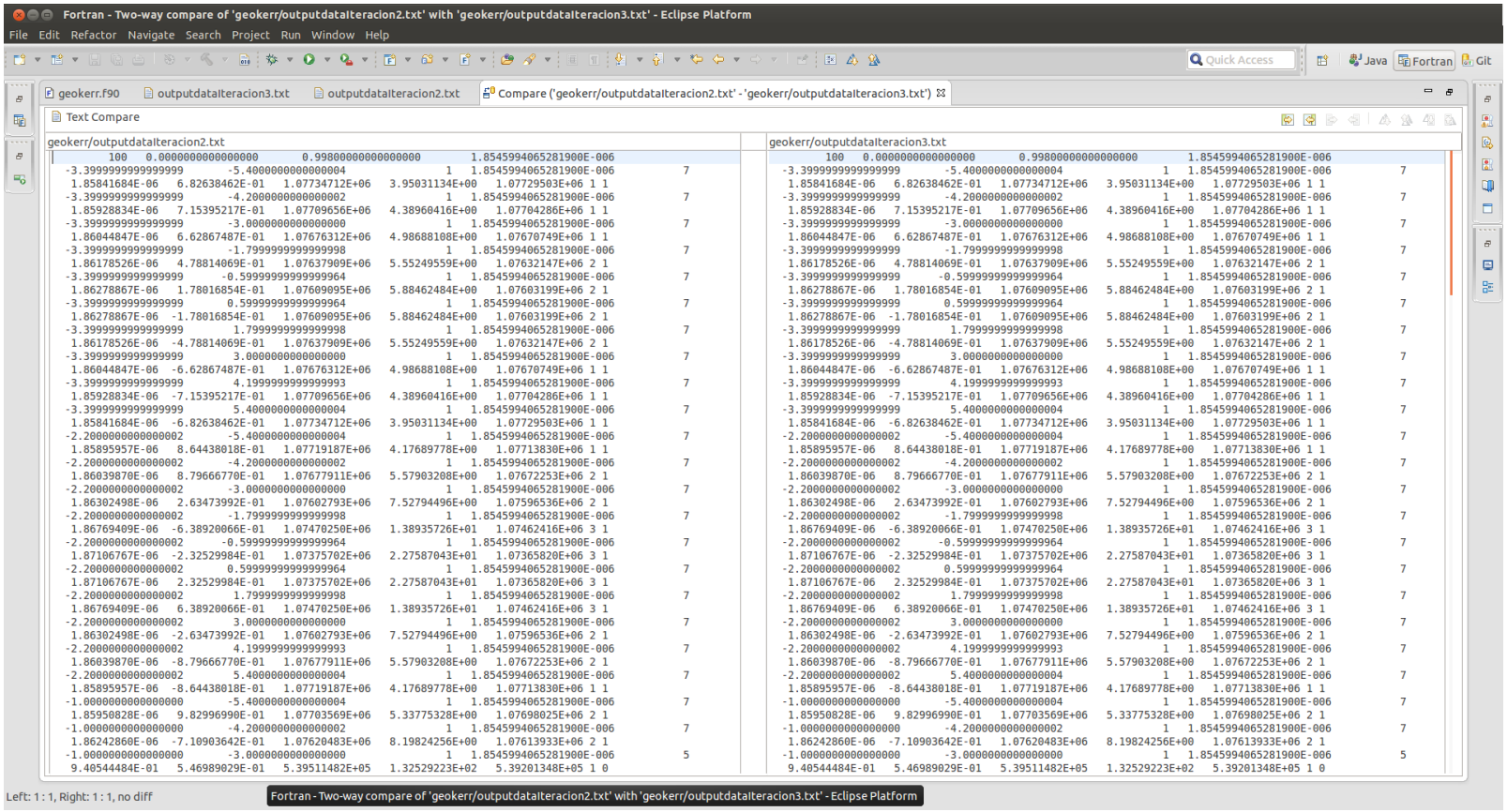


Figura 8.19: Se Comparan los Resultados de las Dos Versiones del Programa Transformado geokerr.f90 Entre la Iteración 2 y la 3

Iteración 4: Eliminar las instrucciones DO escritas en formato antiguo

Al igual que en el ejemplo anterior se modernizará la forma de escritura de las instrucciones DO que no hayan sido terminadas con la correspondiente instrucción END DO. Aplicando el flujo de trabajo propuesto se procede a realizar la iteración.

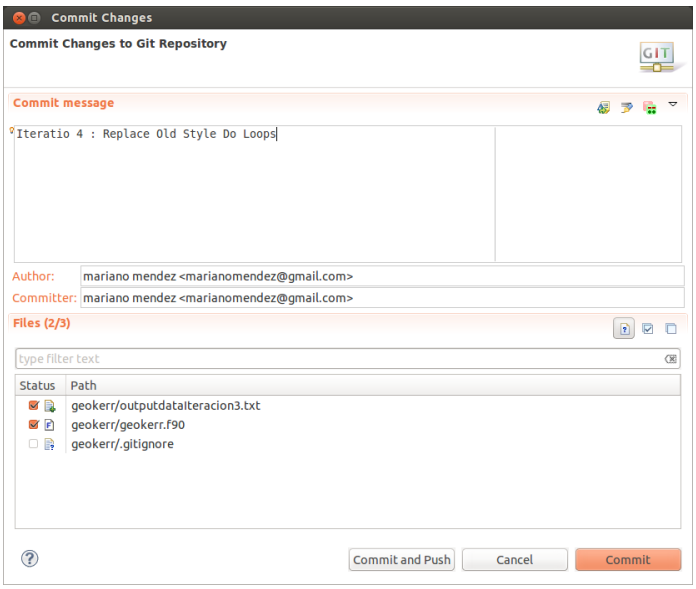


Figura 8.20: Commit de la Versión de la Iteración 3 del Programa al Iniciar la Iteración 4

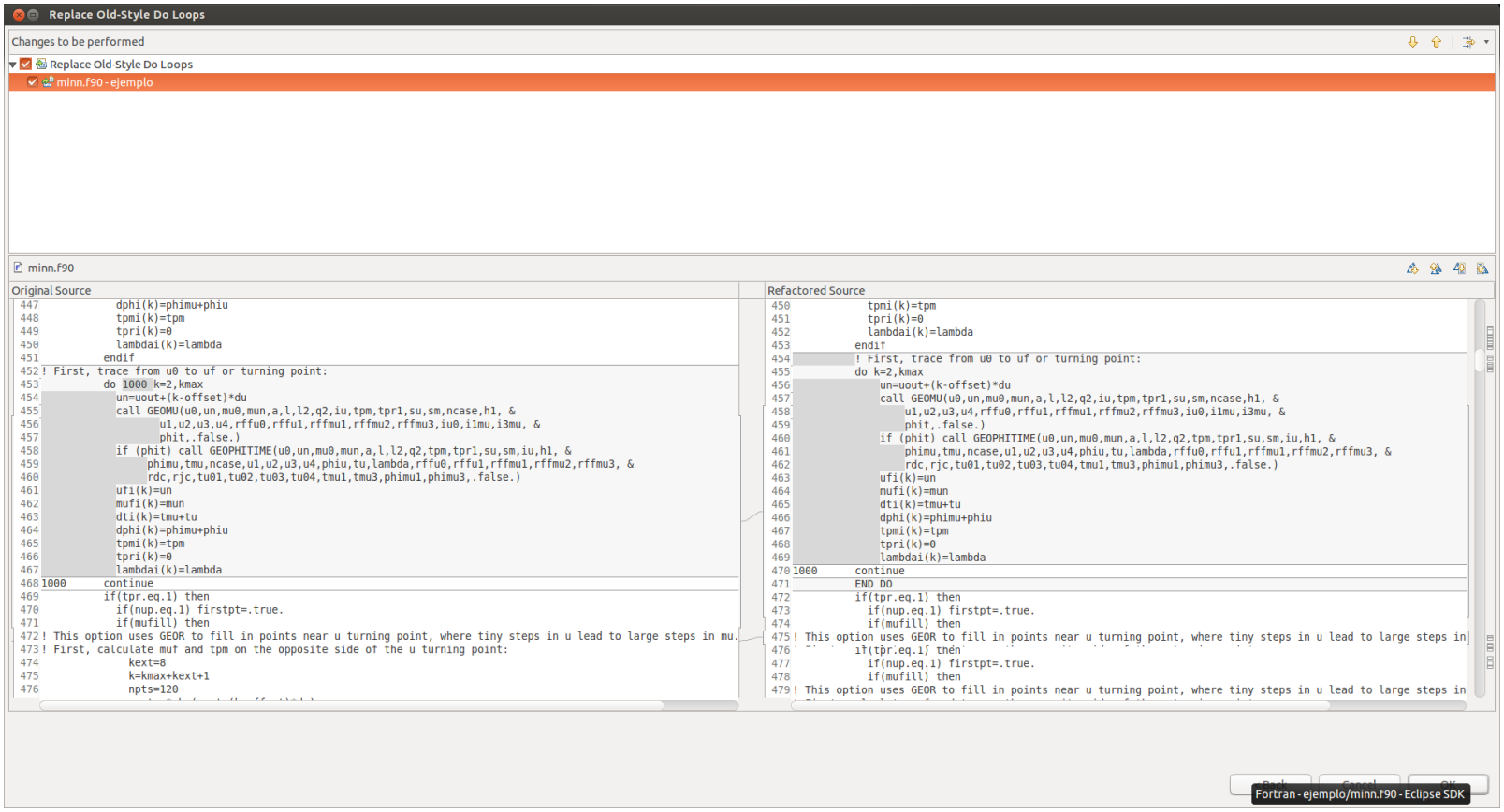


Figura 8.21: Editor del IDE Después de la Aplicación de la Transformación de Código Fuente de la Iteración 4

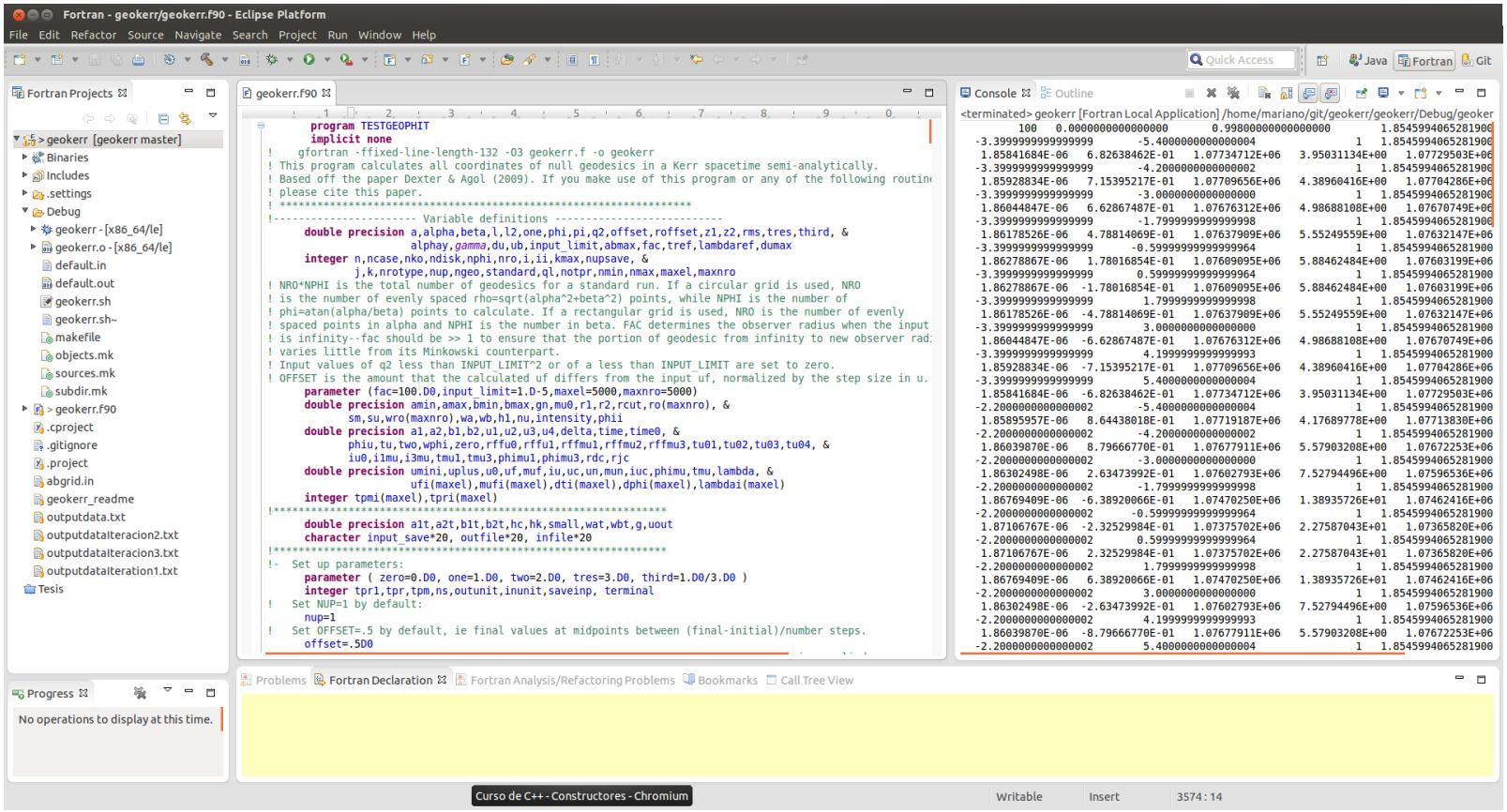


Figura 8.22: Resultados Obtenidos en la Corrida de geokerr.f90 en la Cuarta Iteración

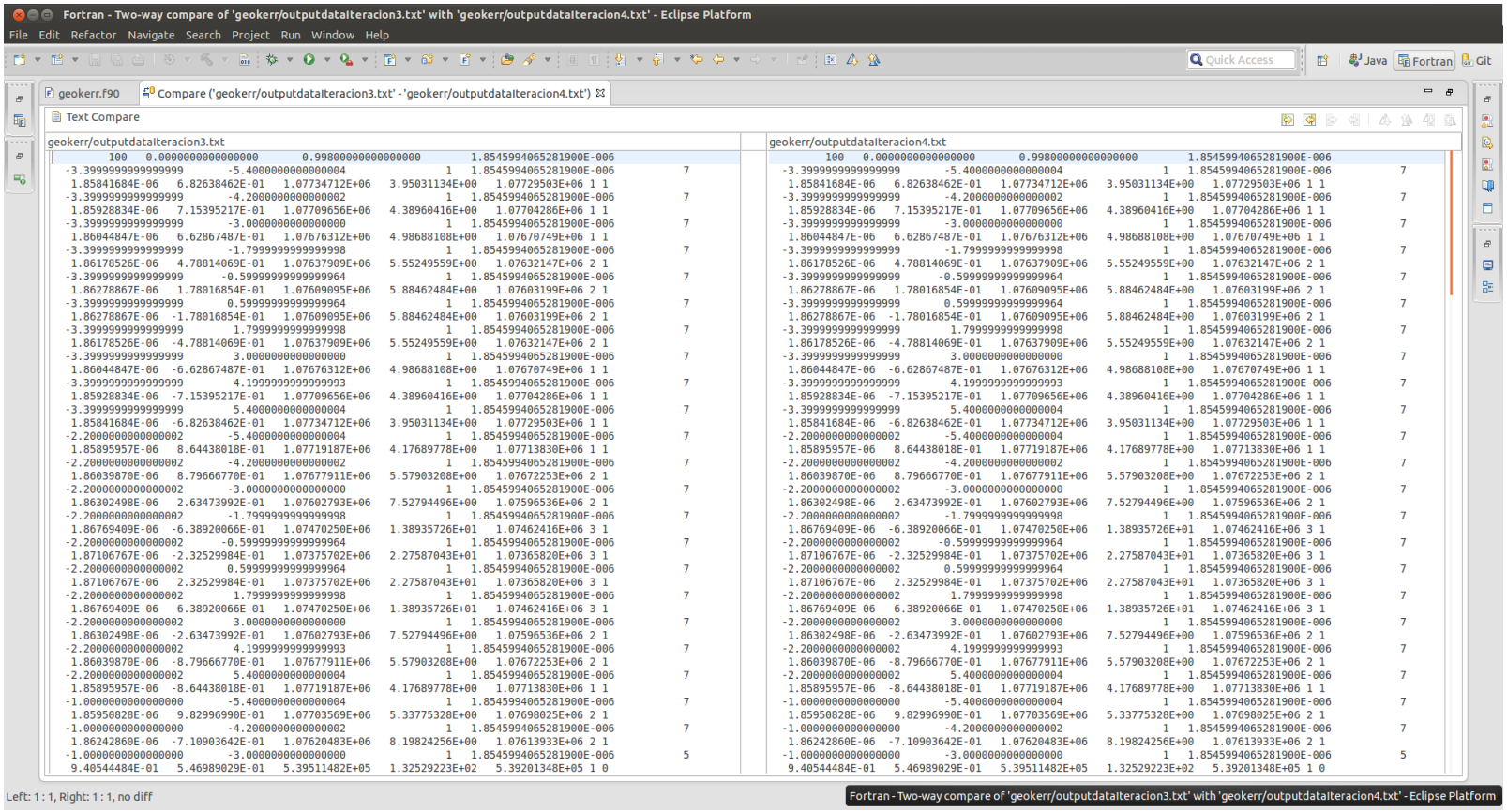


Figura 8.23: Se Comparan los Resultados de las Dos Versiones del Programa Transformado geokerr.f90 Entre la Iteración 3 y la 4

Lista de cambios no realizados y propuestos por la retroalimentación del proceso:

1. introducir Implicit None.
2. eliminar etiquetas no referenciadas.
3. agregar el especificador INTENTIN a las rutinas según corresponda.
4. eliminar as instrucciones no estructuradas como el GOTO.

8.1.1. Iteración 4.1: Elimiar Etiquetas no Referenciadas

Una vez planteada la iteración 4, surge la necesidad de limpiar del código fuente aquellas instrucciones que poseen etiquetas que ya no son más referenciadas. En el caso particular de la instrucción CONTINUE es posible eliminarla completamente. Para acotar la extensión del texto, esta iteración será la iteración 4.1, ya que es un cambio menor íntimamente relacionado con la iteración 4. Esta transformación ha sido desarrollada durante [151], la misma elimina todas aquellas instrucciones CONTINUE que tengan una etiqueta y no sean referenciadas por ninguna instrucción de salto ver Figura 8.28, Figura 8.25, Figura 8.26 y Figura8.25:

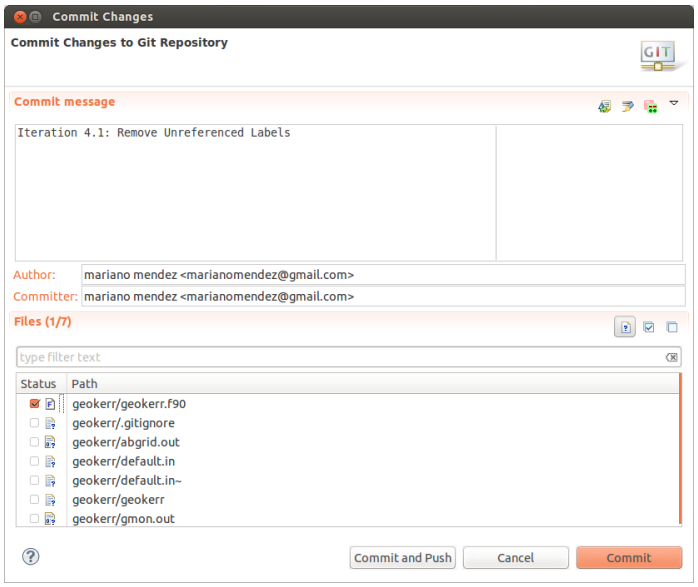


Figura 8.24: Commit de la Versión a Partir de la Iteración 4

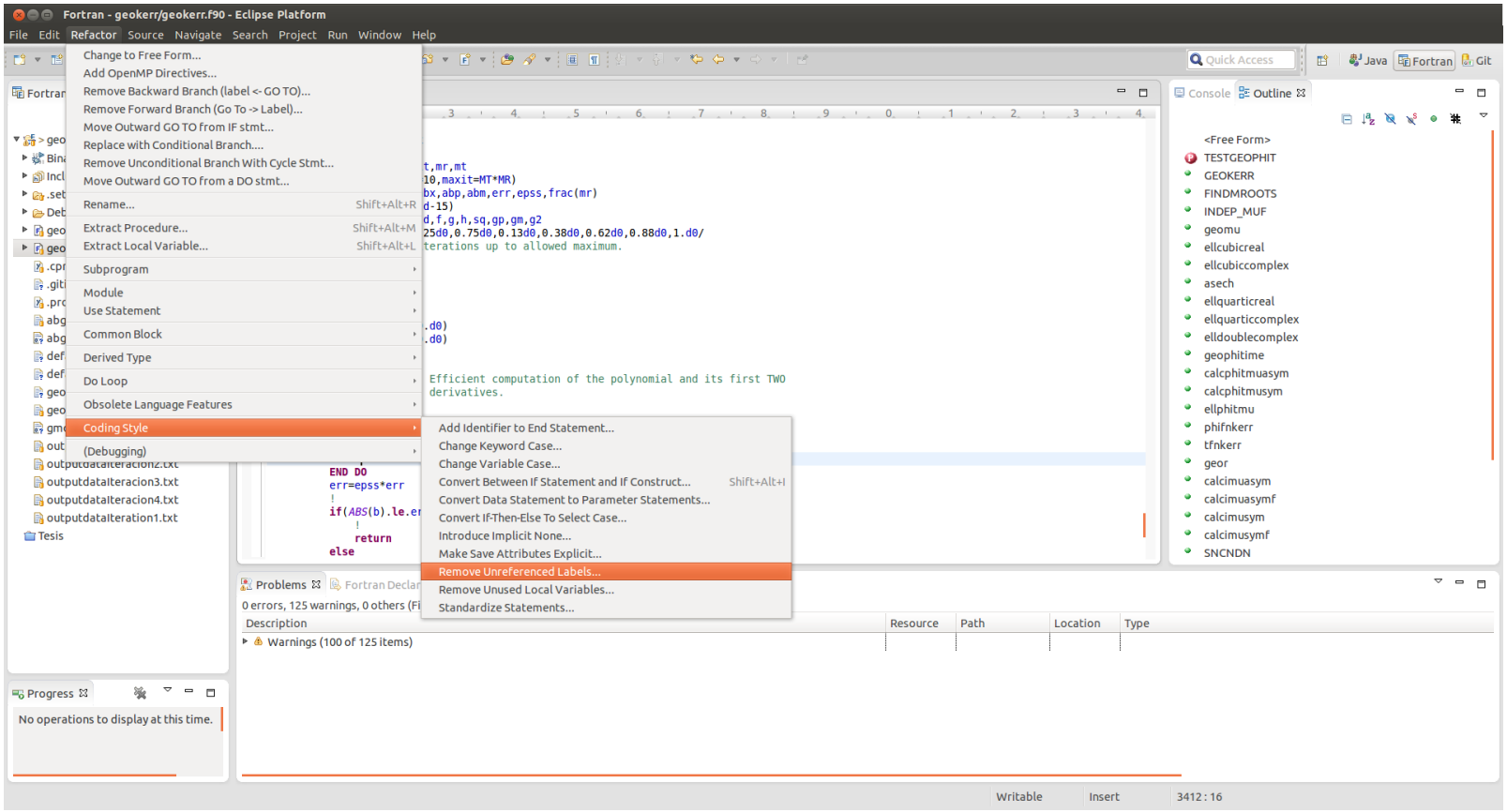


Figura 8.25: Menú de la Transformación de Código Fuente de la Iteración 4.1, Remove Unreferenced Labels

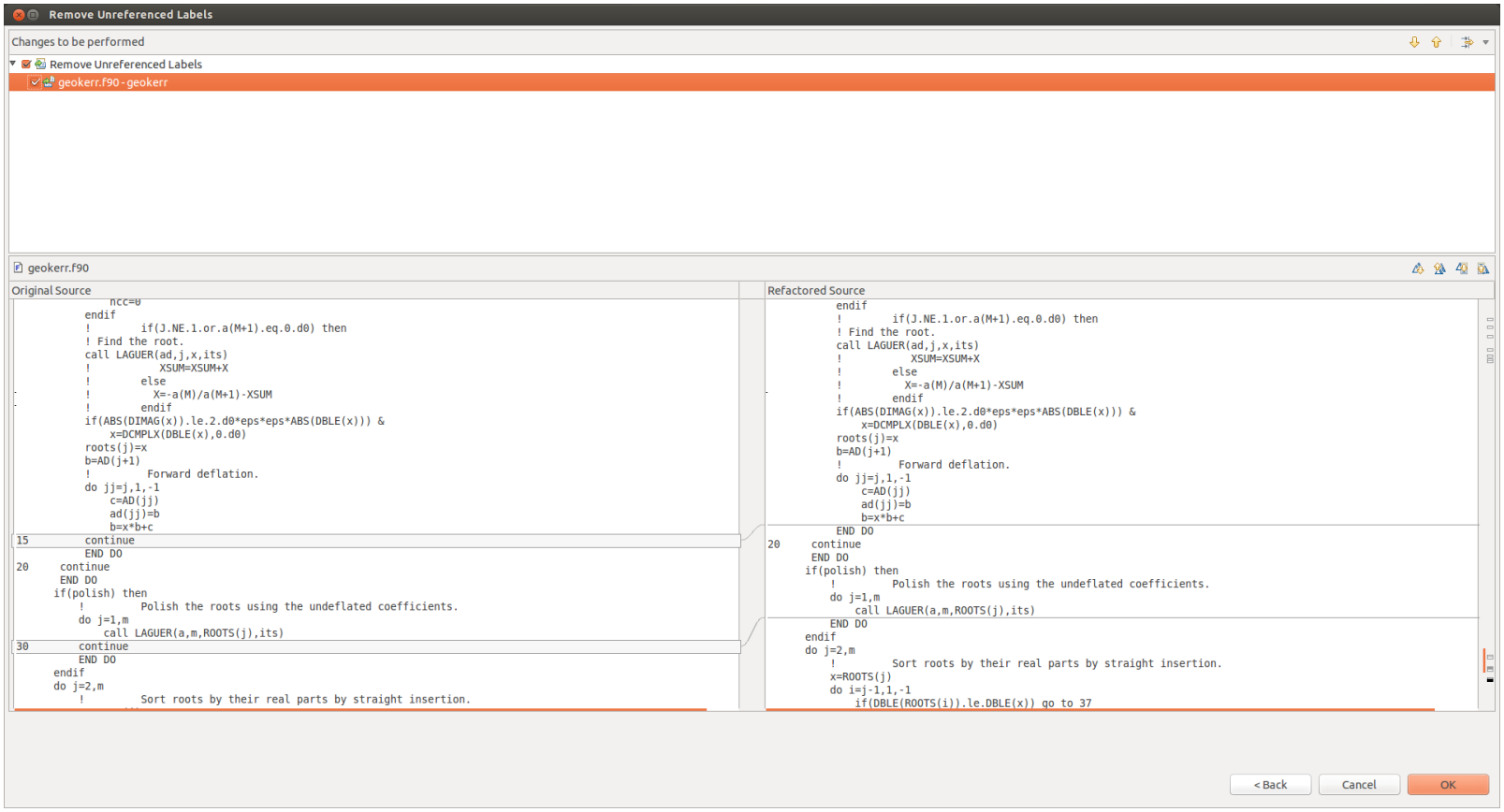


Figura 8.26: Vista de Antes y Después de la Aplicación de la Transformación Sobre el Código Fuente

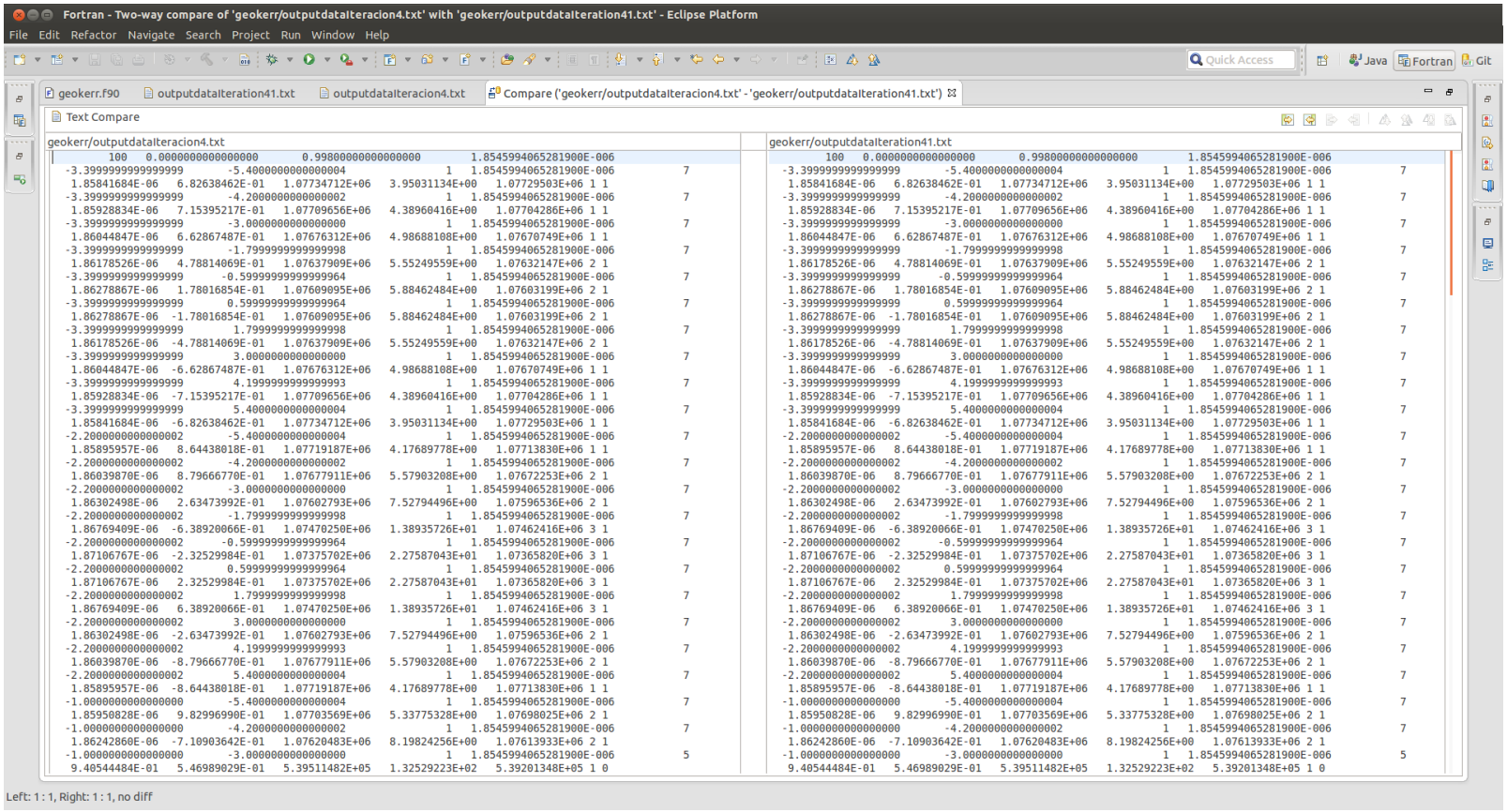


Figura 8.27: Se Comparan los Resultados de las Dos Versiones del Programa Transformado geokerr.f90 Entre la Iteración 4 y la 4.1

```
[h]
Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
40.00      0.06     0.06   482506     0.12     0.12  rc_
20.00      0.09     0.03    44784     0.67     0.67  rd_
13.33      0.11     0.02   118272     0.17     0.17  laguer_
6.67       0.12     0.01    88768     0.11     0.11  rf_
6.67       0.13     0.01    31296     0.32     0.96  ellquarticcomplex_
6.67       0.14     0.01    20000     0.50     3.22  geomu_
6.67       0.15     0.01    20000     0.50     0.50  sncndn_
0.00       0.15     0.00    74352     0.00     0.73  rj_
0.00       0.15     0.00    52624     0.00     0.62  ellquarticreal_
0.00       0.15     0.00    14784     0.00     0.23  calcimusym_
0.00       0.15     0.00    14784     0.00     1.35  zroots_
0.00       0.15     0.00    10000     0.00     4.20  calcphitmusym_
0.00       0.15     0.00    10000     0.00    15.00  geokerr_
0.00       0.15     0.00    10000     0.00     8.56  geophitime_
```

Figura 8.28: Salida del perfilado obtenida con gprof

Una vez aplicado este cambio, se procede a iniciar el ciclo de la iteración 5.

8.2. Iteración 5: Paralelización

El enfoque utilizado en la siguiente iteración es el mismo que en las iteraciones anteriores con la diferencia que para poder aplicar las transformaciones de paralelización, en primer lugar es necesario saber en qué parte del código fuente es factible realizarlo. Para ello, en primer lugar para reconocer los posibles cuellos de botella de procesamiento secuencial se debe realizar un perfilado de la aplicación en tiempo de ejecución de la misma. Para obtener el perfilado se ha compilado el programa con la opción -pg, que permite obtener el perfilado en tiempo de ejecución del programa con la herramienta gprof. Cabe destacar que el tool chain utilizado es el de gnu, gfortran v-4.8.

Del análisis del mismo se desprende que más del 73 % de la ejecución del programa es consumida por tres rutinas: rc(), rd(), laguer() (el perfilado completo puede ser observado en B.2.1). Estas tres rutinas, en ese mismo orden, son las que

más cómputo insumen a lo largo de la ejecución del mismo. Teniendo en cuenta que las dos primeras no poseen ninguna instrucción DO aquellas que por su características son las ideales para paralelizar con OpenMP, se intentará paralelizar la rutina laguer() que posee dos ciclos DO.

8.2.1. Instrucción DO a Paralelizar

La instrucción DO que se intentará paralelizar se lista a continuación:

```
its=iter
b=A(m+1)
err=ABS(b)
d=DCMPLX(0.d0,0.d0)
f=DCMPLX(0.d0,0.d0)
abx=ABS(x)
do j=m,1,-1
!    Efficient computation of the polynomial and its first TWO
!    derivatives.
     f=x*f+d
     d=x*d+b
     b=x*b+A(j)
     err=ABS(b)+abx*err
END DO
```

Dentro de los pasos necesarios para realizar dicha transformación se realizan distintos chequeos para determinar si una instrucción DO es paralelizable y en estos chequeos todas aquella variables candidatas a ser utilizadas en una cláusula REDUCTION. La idea de la cláusula REDUCTION es la de crear una copia privada por thread de una variable que es utilizada como se muestra en el cuadro:

x = x operator expr x = expr operator x (except subtraction) x = intrinsic(x, expr) x = intrinsic(expr, x)
x is a scalar variable in the list expr is a scalar expression that does not reference x intrinsic is one of MAX, MIN, IAND, IOR, IEOR operator is one of +, *, -, .AND., .OR., .EQV., .NEQV.

Cabe destacar que la cantidad de chequeos necesaria para la determinación de la posibilidad de paralelización es muy compleja. En este ejemplo se ha utilizado uno muy sencillo para ilustrar la posibilidad de la aplicación del mismo en un caso de la vida real, en [203, 201] se tratan con más detalles este tipo de transformaciones. La cláusula REDUCTION es una mezcla entre las clausulas private, shared y atomic. Ésta permite actuar como un acumulador a una variable compartida (shared) sin la utilización de la cláusula atomic, teniendo en cuenta que se debe especificar el tipo de acumulación que se realiza. Además cabe destacar que según el operador la variable debe ser pre-inicializada en ciertos valores que son:

- Cero si las operaciones son: $+, -, |,^{,} ||$
- Uno si las operaciones son: $*, \&\&$

Para el caso en cuestión se selecciona la instrucción DO que se desea paralelizar, y si ésta es paralelizable se mostrará la vista de Diff con el código fuente. En este caso la transformación ha detectado que este DO no es paralelizable porque una de las variables que debiera llevar la cláusula reduction no se encuentra inicializada en su valor correspondiente, esta variable es b, ver Figura 8.29.

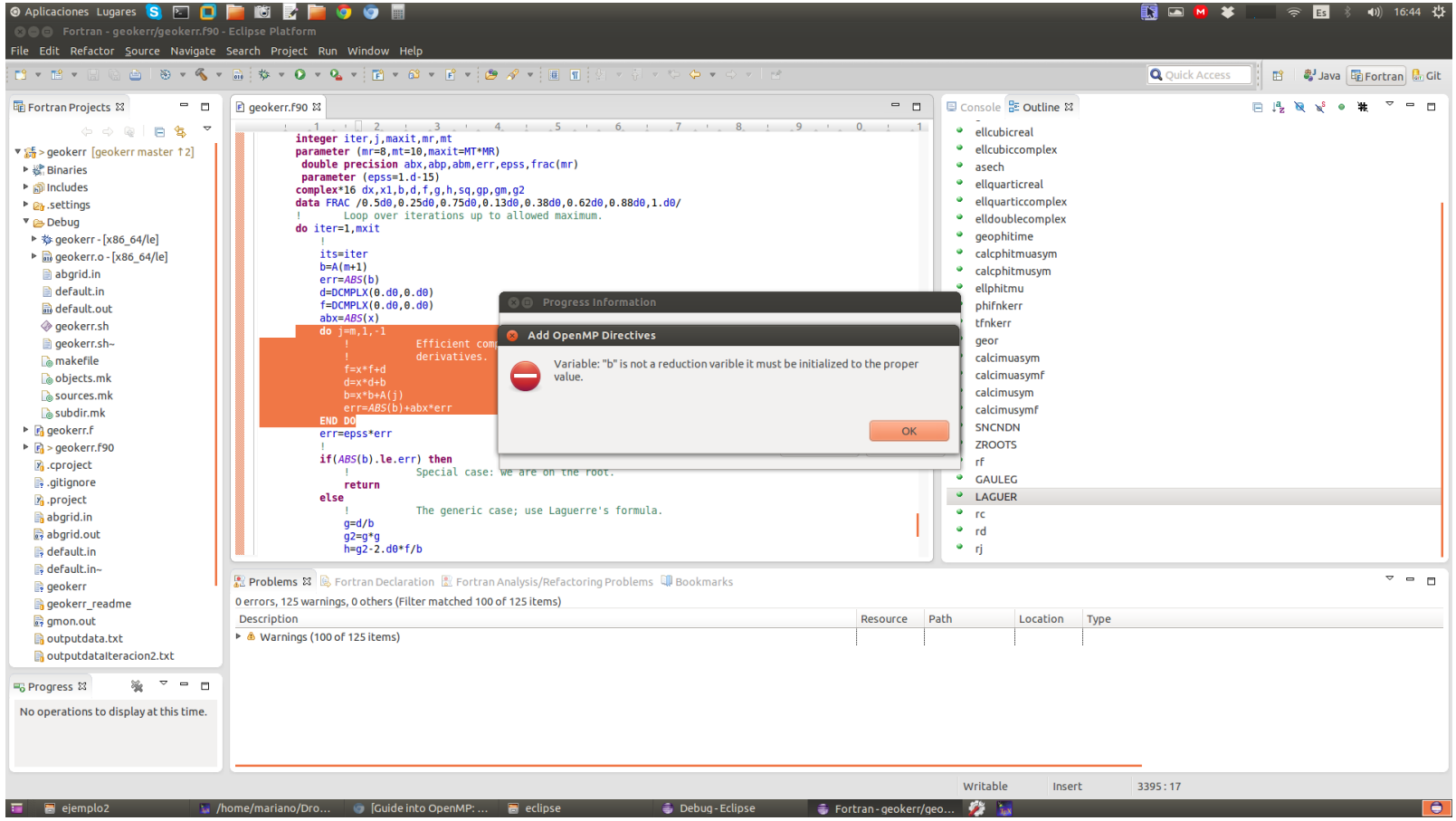


Figura 8.29: Se Comparan los Resultados de las Dos Versiones del Programa Transformado geokerr.f90 Entre la Iteración 4 y la 4.1

En este caso no es posible paralelizar el ciclo utilizando OpenMP. Cabe destacar que el ciclo más externo tampoco es paralelizable pues la cantidad de iteraciones (se desprende del código que son unas 80 iteraciones) es tan pequeña que no resultaría de valor agregado.

Visto y considerando que no hay más rutinas a las que se le pueda hacer algún aporte respecto a mejoras de rendimiento utilizando OpenMp, la versión del código fijada al final de la iteración 4.1 es aquella que se tendrá como referencia.

8.3. Resumen

En este capítulo se ha aplicado nuevamente el proceso de desarrollo dirigido por el cambio y el flujo de trabajo propuesto en cada una de las cuatro iteraciones en las que se ha actualizado el programa, escrito por científicos y utilizado como herramienta en una publicación científica, ver [64]. El ejemplo, escrito inicialmente en FORTRAN 77, ha sido transformado mediante la aplicación de pequeñas mejoras al código fuente a una versión más moderna del lenguaje como es Fortran 90.

Capítulo 9

Aplicación del Proceso para Paralelización: Caso de estudio 3 y 4

En este capítulo se estudiará una tercera aplicación del proceso detalladamente en cada uno de ellos se mostrarán las fases del proceso y el flujo de trabajo propuestos el los capítulos anteriores. El enfoque que se utilizará en este capitulo está dirigido exclusivamente hacia la paralelización de código fuente utilizando OpenMP como los presentados en [203, 201].

9.1. Caso de Estudio 3: Prime Numbers

El siguiente ejemplo fue extraído de `http://people.sc.fsu.edu/~jburkardt/f77_src/prime_serial/prime_serial.html` y simplemente consiste en un algoritmo escrito en FORTRAN 77 capaz de contar los primos dentro de los N números enteros positivos. Si bien el programa puede considerarse trivial, la idea es lograr aplicar el Desarrollo Guiado por el Cambio para paralelizar. Inicialmente se comienza con una sola iteración cuyo objetivo es obtener el primer cambio: el programa serie funcionando. En la etapa de retroalimentación surgirán, posi-

blemente, nuevas iteraciones dado que el proceso mismo de paralelización es un cambio iterativo e incremental.

9.1.1. Iteración 1: Instalación y Perfilado

Inicialmente se procede a realizar el primer cambio cuyo resultado es el programa funcionando y el perfilado completo. Para ello se procederá a realizar el flujo de trabajo propuesto:

1. Establecer una versión inicial del código fuente

2. Transformar el código fuente

3. Verificar el código fuente obtenido

4. Validar los resultados numéricos

5. Aceptar/Rechazar el cambio en base a los resultados numéricos

6. Documentar

La versión del código fuente se obtiene de `http://people.sc.fsu.edu/~jburkardt/f77_src/prime_serial/prime_serial.html`, se utiliza Eclipse Photran 8 como herramienta de desarrollo, PhotranLint plug-in para Photran desarrollado en este trabajo, el compilador que se utiliza es gfortran versión 4.8. Como software de control de versionado se utiliza git y su plug-in para Eclipse. La iteración completa puede ser vista en la Figura 9.1.

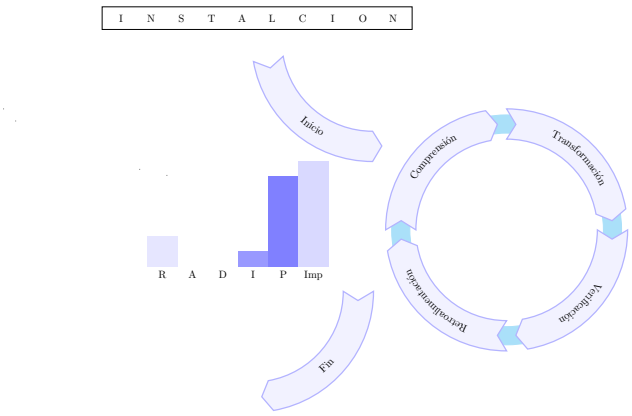


Figura 9.1: Proceso de Desarrollo Dirigido por el Cambio Primera Iteración Caso de Estudio Prime Numbers.

En esta etapa se procede a crear un proyecto Fortran en la herramienta Photran, se pone bajo gestión de versionado el proyecto y los fuentes originales, ver Figura 9.2. A continuación se habilitan la infraestructura de refactorización y el análisis de código fuente Fortran de Photran, ver Figura 9.3, Figura 9.4 y Figura 9.5. Posteriormente se setean algunos parámetros de compilación y linkedición como por ejemplo la opción -pg para que se realice el perfilado de la ejecución del programa.

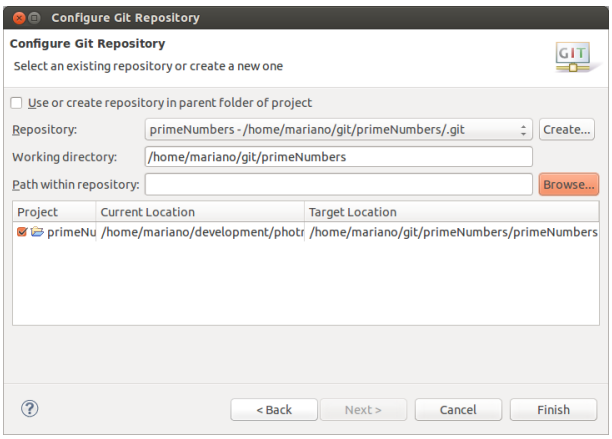


Figura 9.2: Prime Numbers Control de Versiones

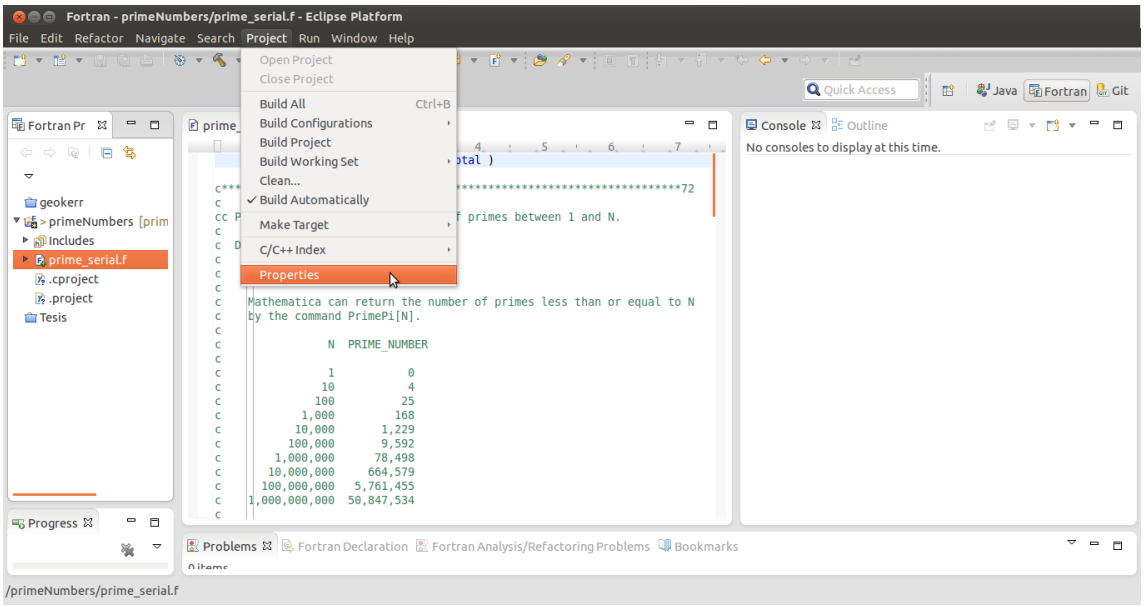


Figura 9.3: Activación de la Infraestructura de Refactorización

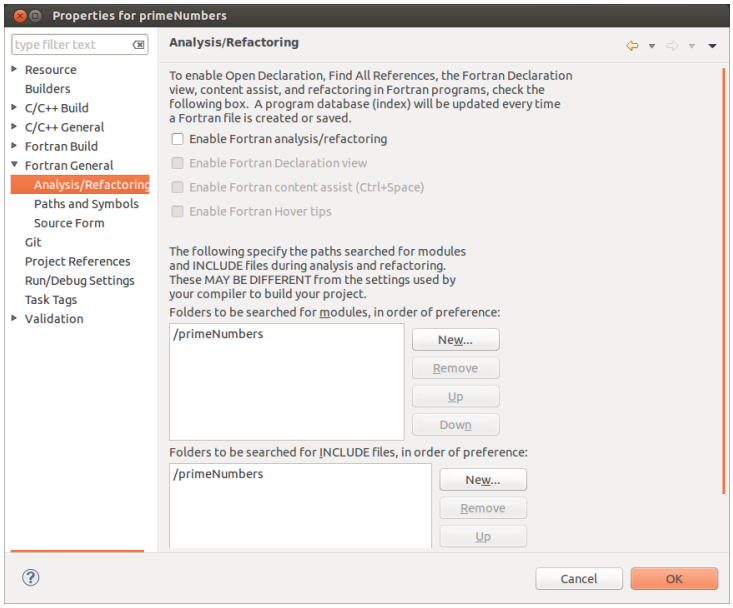


Figura 9.4: Activación de la Infraestructura de Refactorización

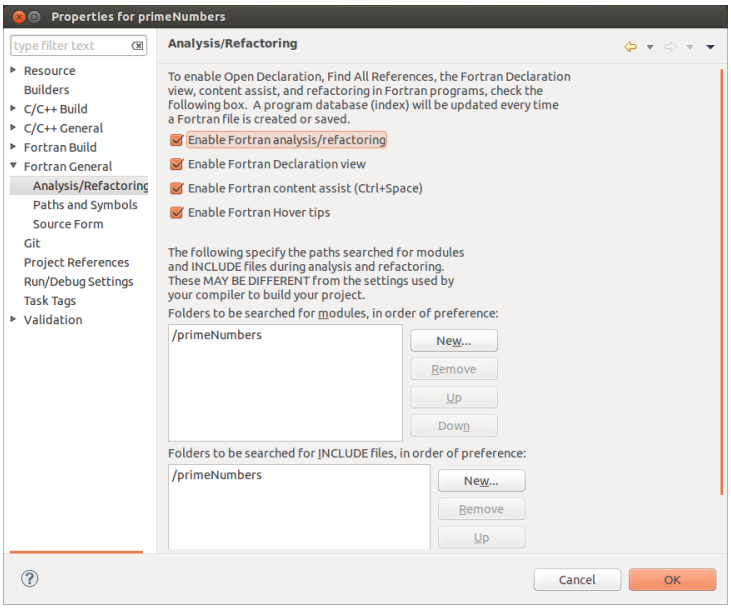


Figura 9.5: Activación de la Infraestructura de Refactorización

Una vez seteadas todas las variables y parámetros de configuración se procede a la ejecución y perfilado del programa, ver Figura 9.6. Debido a que los autores del programa han proporcionado un archivo con la salida de la ejecución del mismo se realiza la verificación numérica mediante la comparación de los datos obtenidos en dicha ejecución contra aquella obtenida por los autores, ver Figura 9.7. Puede observarse que los valores numéricos obtenidos son iguales, la única y esperable variación son los tiempos de cómputo calculados en cada corrida, dado que cada una ha sido ejecutada en máquinas distintas.

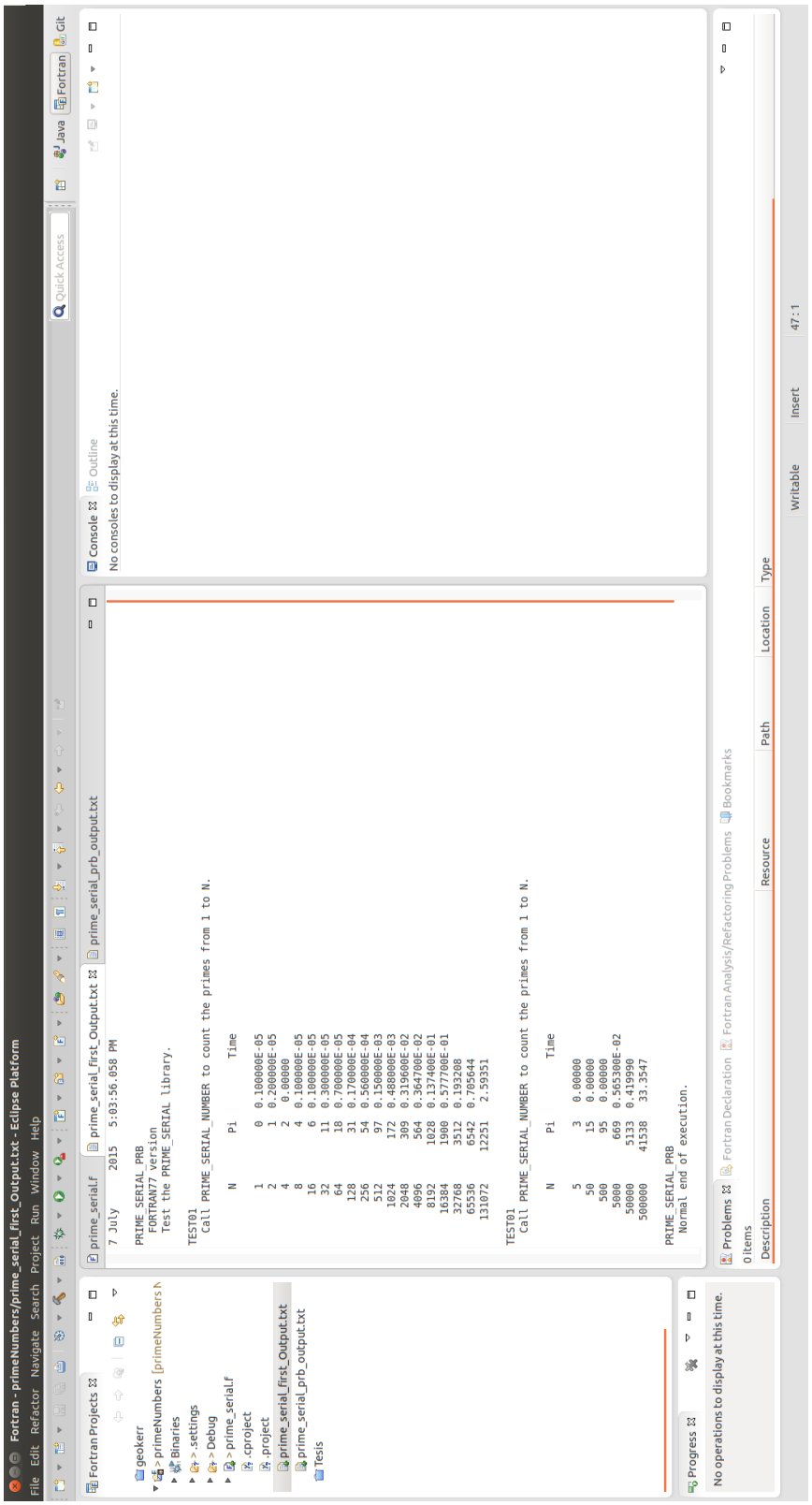


Figura 9.6: Obtención de los Resultados de la Ejecución de PrimeNumbers.f

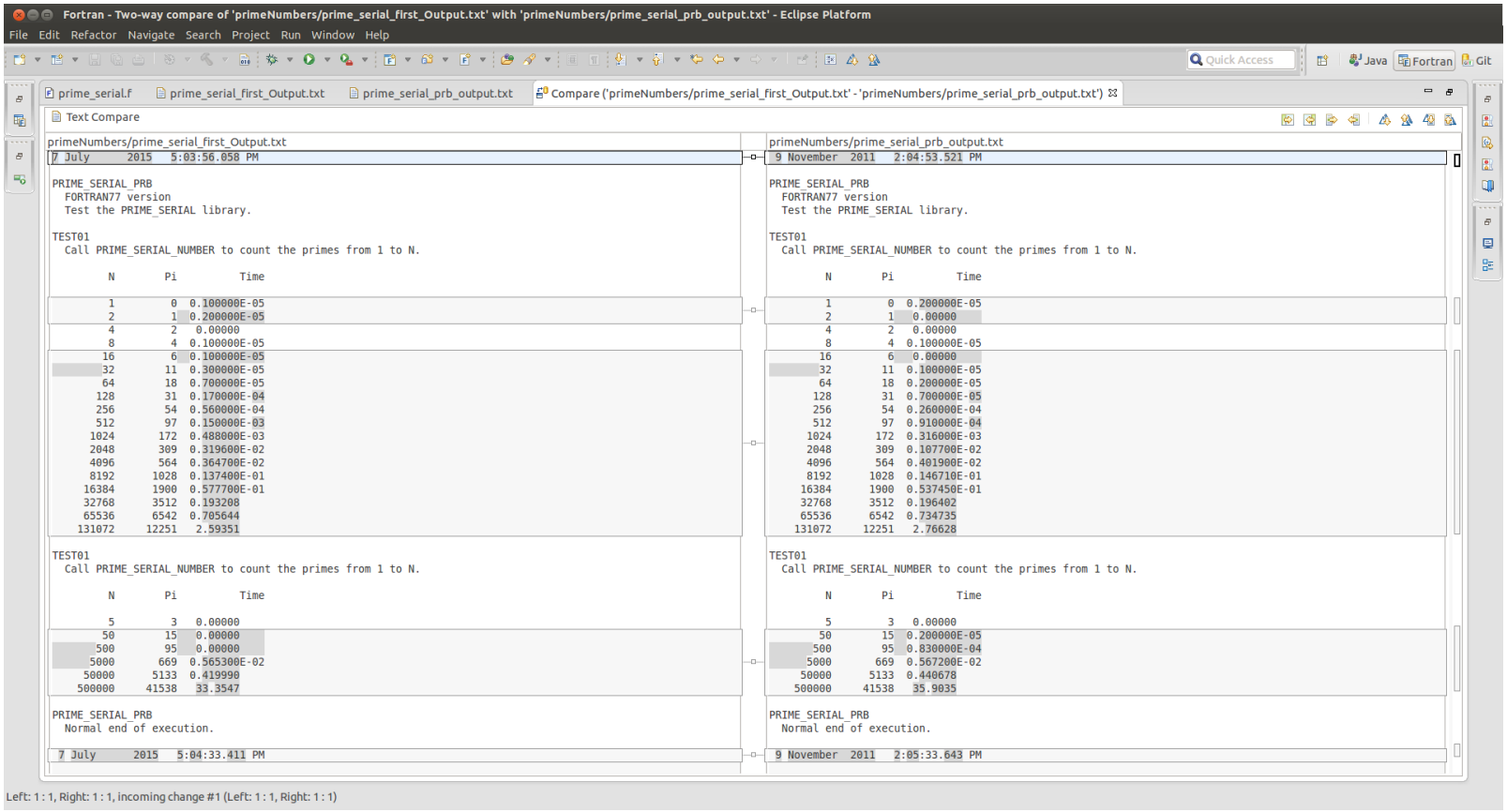


Figura 9.7: Comparación de Salidas de Ejecución

A partir del análisis de los datos obtenidos, se puede asegurar que el programa está corriendo correctamente, respecto a los valores numéricos. A continuación se realiza un análisis del perfilado de ejecución de la aplicación:

```
granularity: each sample hit covers 2 byte(s) for 0.02% of 42.20 seconds


index % time    self  children    called     name
42.20    0.00      24/24          prime_number_sweep_ [2]
[1]    100.0   42.20    0.00      24         prime_number_ [1]
-----------------------------------------------
0.00   42.20       2/2           MAIN__ [3]
[2]    100.0    0.00   42.20       2         prime_number_sweep_ [2]
42.20    0.00      24/24          prime_number_ [1]
-----------------------------------------------
0.00   42.20       1/1           main [4]
[3]    100.0    0.00   42.20       1         MAIN__ [3]
0.00   42.20       2/2           prime_number_sweep_ [2]
0.00    0.00       2/2           timestamp_ [5]
-----------------------------------------------
<spontaneous>
[4]    100.0    0.00   42.20                 main [4]
0.00   42.20       1/1           MAIN__ [3]
-----------------------------------------------
0.00    0.00       2/2           MAIN__ [3]
[5]      0.0    0.00    0.00       2         timestamp_ [5]
-----------------------------------------------
```

Como era de esperar la rutina prime_number_sweep() es aquella que genera la mayor parte del cómputo. Tras realizar un análisis de la misma se decide en la etapa de retroalimentación generar dos nuevos cambios que conllevan a dos nuevas iteraciones, ver Figura 9.8:

- Cambio 2: Cambiar a formato Libre

- Cambio 3: Aplicar la transformación de paralelización en la rutina prime_number_sweep()

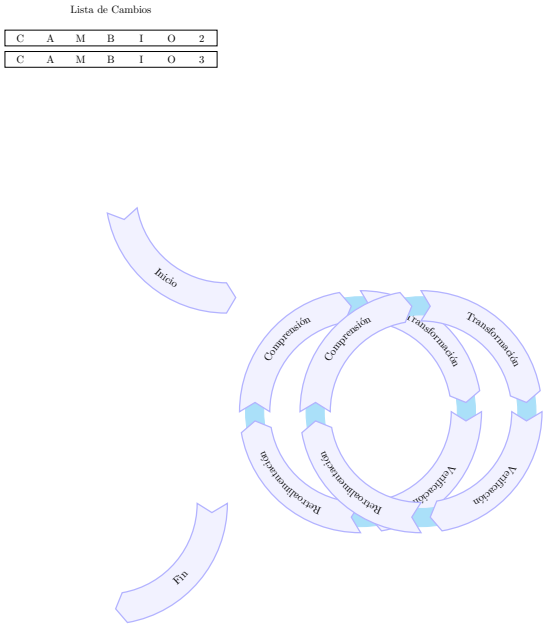


Figura 9.8: Dos Iteraciones Agregadas al Proceso en la Etapa de Retroalimentación de la Iteración 1

9.1.2. Iteración 2: Cambiar a formato libre

Al igual que en los casos de estudio anteriores en los Capítulos 7 y 8 se procede, a fijar la versión de inicio de la iteración antes de realizar el cambio en cuestión, ver Figura 9.9. En este caso es eliminar el formato fijo del código fuente del programa. Para ello se utiliza una transformación automática de código fuente implementada como una refactorización denominada Change to Free Form, ver Figura 9.10. Una vez seleccionada la entrada del Menú, se despliega una vista de Diff posteriormente a realizar una serie de chequeos necesarios para ejecutar la transformación, por ejemplo que el recurso seleccionado sea un archivo de código fuente Fortran escrito en formato fijo, ver Figura 9.11. Una vez que los cambios son aprobados por el usuario se procede a la compilación, ejecución y posterior comparación de resultados numéricos de salida, ver Figura 9.12, Figura 9.13, Fi-

gura 9.14 y Figura 9.15. De la comparación de ambas ejecuciones se desprende que los resultados numéricos no han variado.

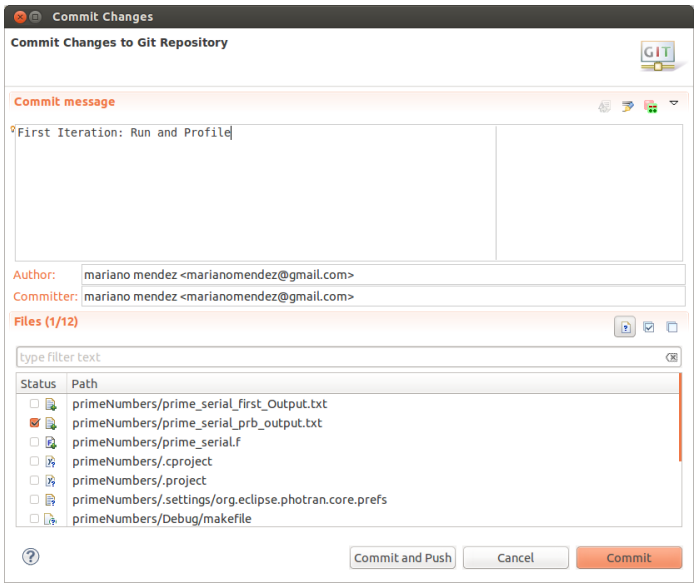


Figura 9.9: Establecimiento de la Versión de Trabajo, Bajo Control de Versiones en Git

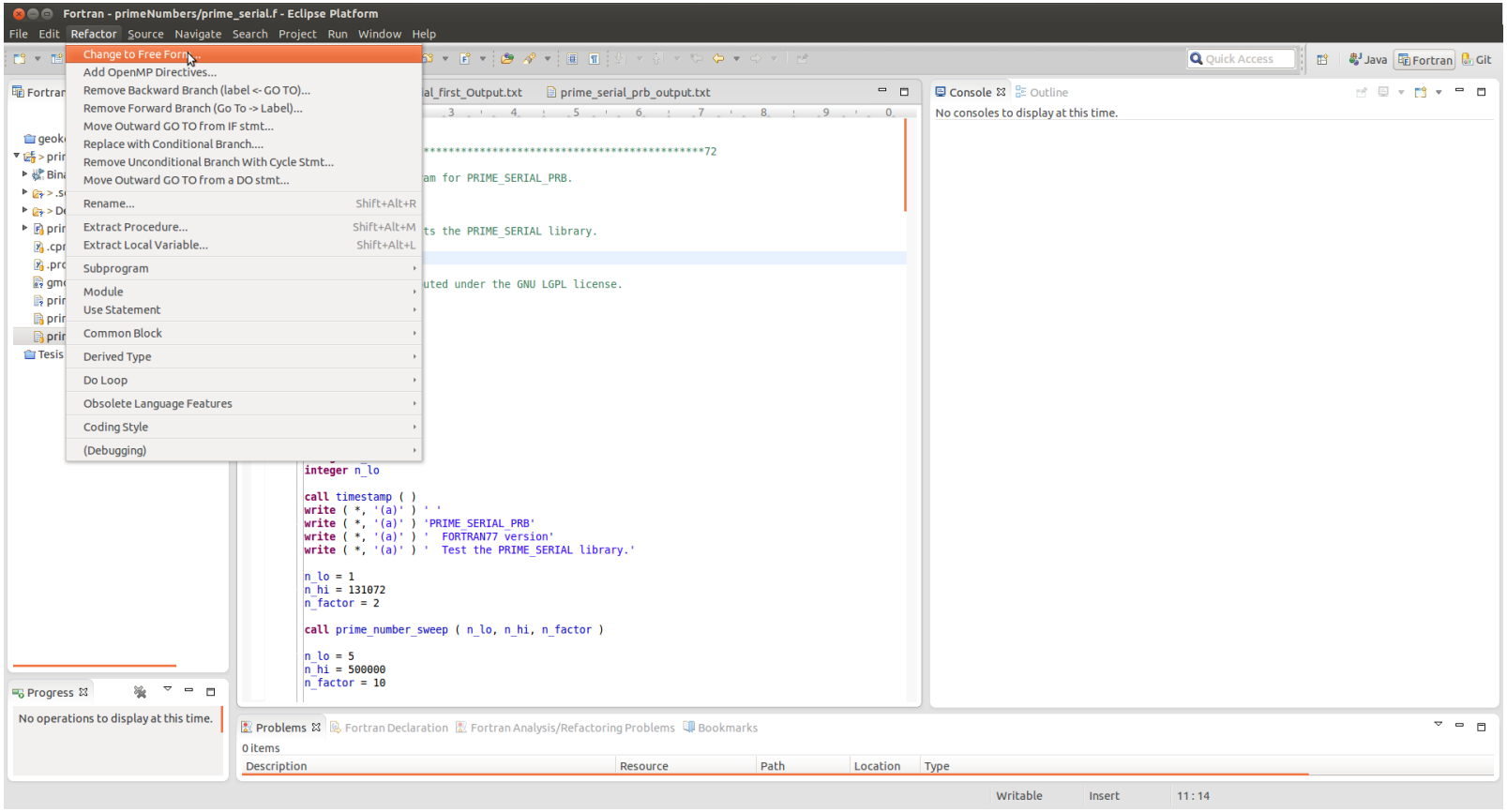


Figura 9.10: Dos Iteraciones Agregadas al Proceso en la Etapa de Retroalimentación de la Iteración 1

Figura 9.11: Vista de Diff Donde se Aprecia el Código Fuente Sin Transformar a la Izquierda de la Pantalla y el Código Fuente Transformado a la Derecha de la Misma

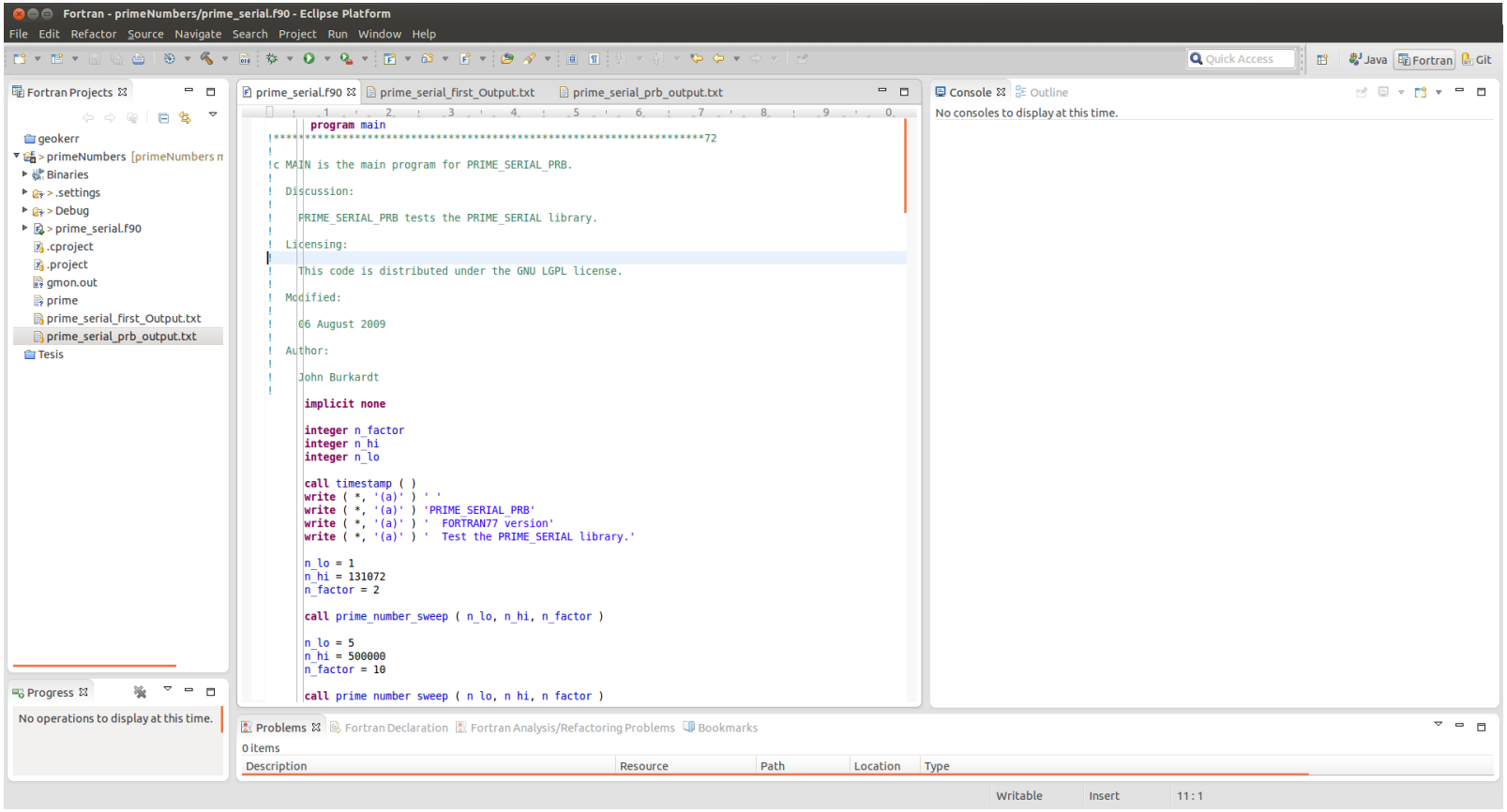


Figura 9.12: Código Fuente ya Transformado

Figura 9.13: Ejecución del Programa

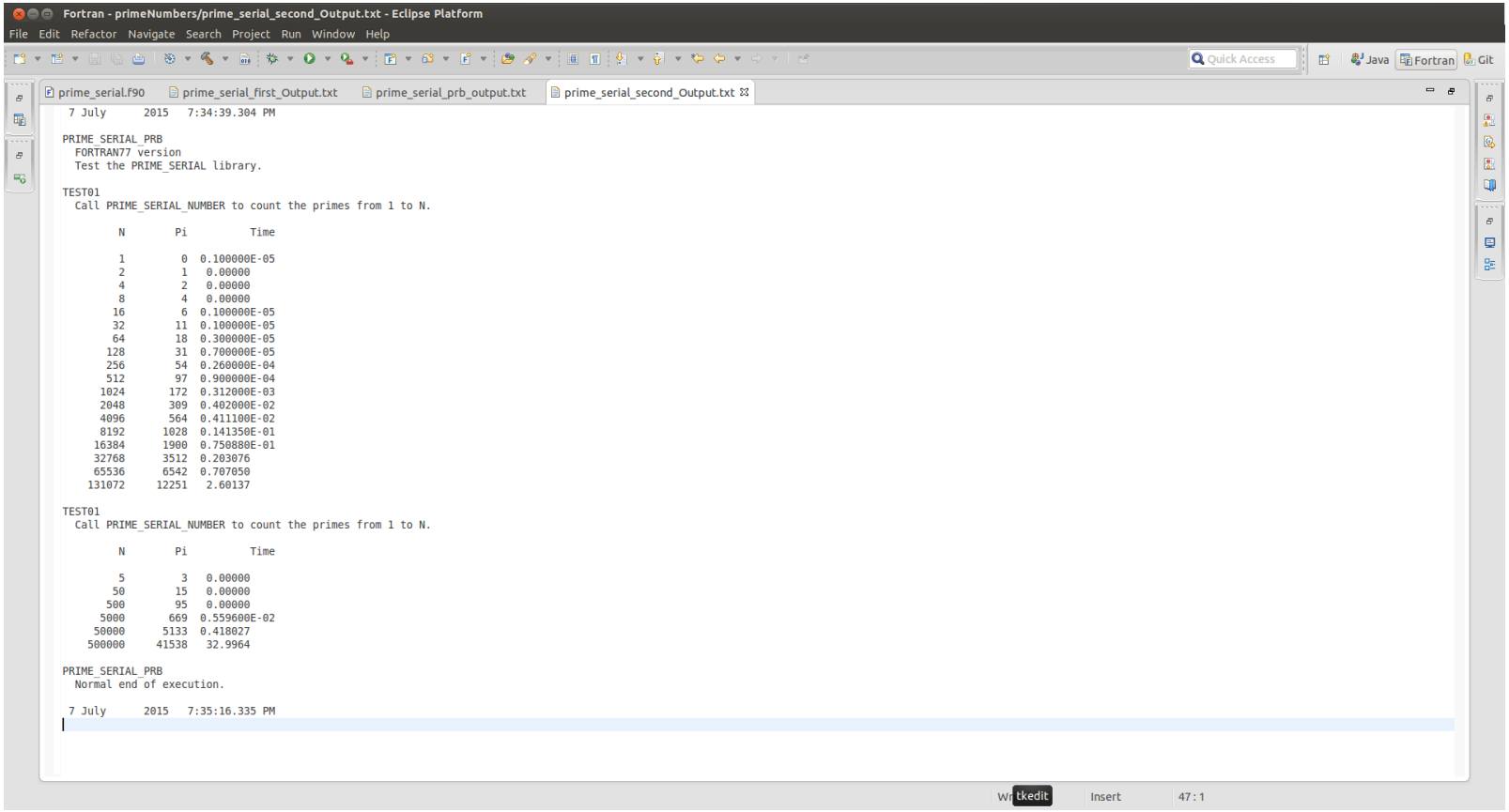


Figura 9.14: Vista de los Resultados Arrojados por la Ejecución

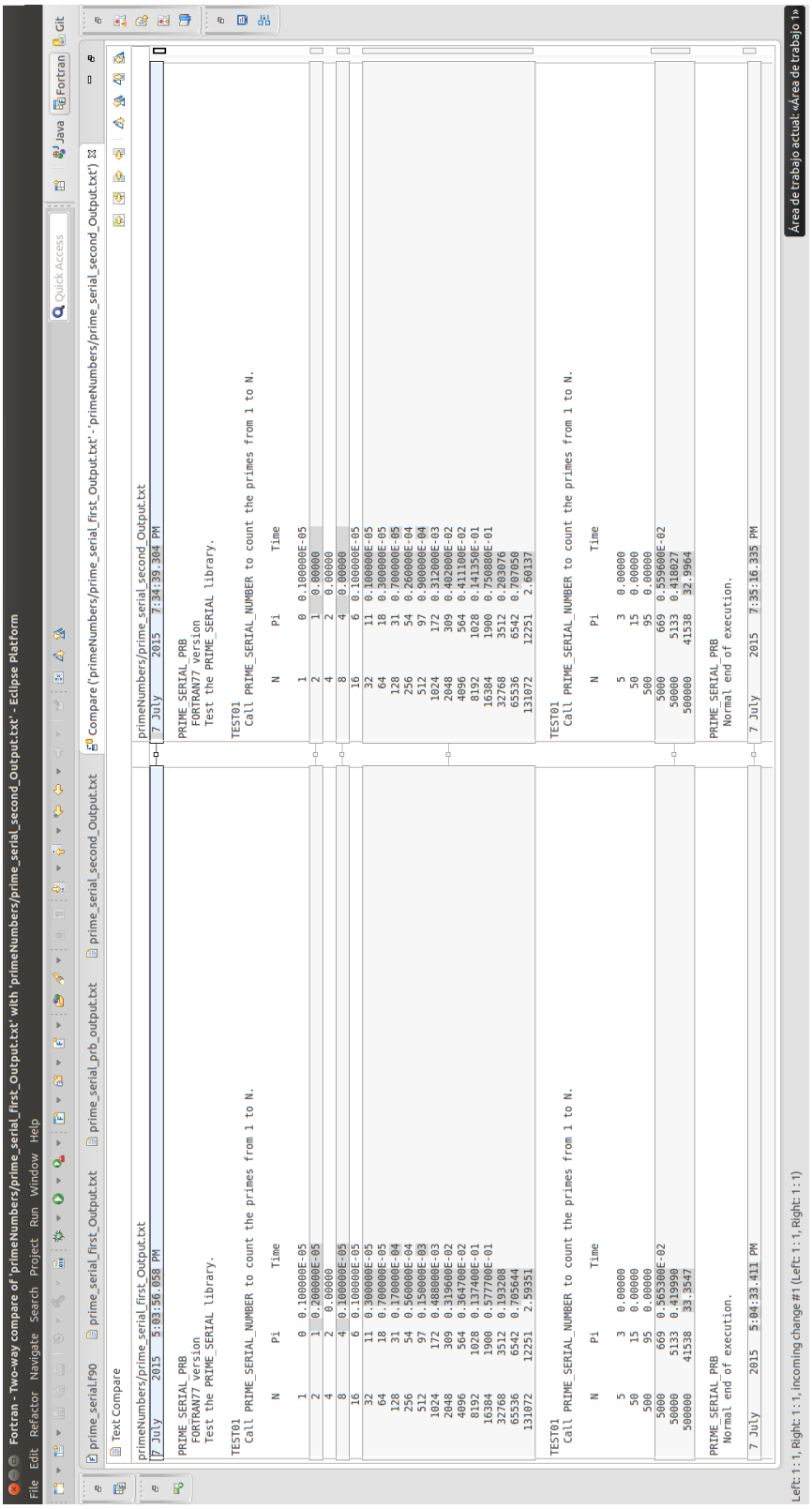


Figura 9.15: Comparación de los Resultados Entre las Salidas de las Dos Versiones del Programa

9.1.3. Iteración 3: Paralelización

Al igual que en la iteración anterior se pone la última versión del código fuente bajo control de versiones, en este caso Git, ver Figura 9.16, y se prosigue al estudio del perfilado para determinar en qué rutina se debería intentar paralelizar el programa. Del perfilado obtenido en la ejecución del programa en la primera iteración se desprende que la rutina prime_number_sweep() es aquella que más cómputo consume dentro de la ejecución del mismo. A partir de este análisis, de la etapa de comprensión del proceso, se aplica la transformación que consiste en la implementación de una refactorización que realiza los chequeos necesarios para determinar si una porción de código fuente Fortran es paralelizable utilizando las directivas de OpenMp. En el caso de cumplir con las condiciones, la transformación se lleva a cabo mediante la construcción de las directivas de OpenMp correspondientes en las cueles se detectarán aquellas variables que sean shared, private o reduction si las hubiera, ver Figura 9.17. Una vez que el usuario acepta los cambios propuestos, el código fuente es transformado y esa transformación se hace visible en el editor del IDE, ver Figura 9.18. Una vez realizado esto, una serie de parámetros tienen que ser agregados a la cadena de compilación y linkedición, como por ejemplo la opción que habilita la utilización de OpenMp en gfortran (-fopenmp) y la biblioteca para ser linkeditada (-lgomp), ver Figura 9.19 y 9.20. Finalmente en la Figura 9.21 se puede ver la ejecución en paralelo del programa. Cabe destacar que si bien los valores numéricos son idénticos que en la versión de la iteración anterior los valores de tiempo de cómputo no han mejorado respecto de la versión serie, ésta se llevó a cabo en un equipo de escritorio cuyo procesador es un Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz de 2 núcleos 4 threads, los resultados de la ejecución se pueden ver en la Figura 9.22. Para determinar si se pudiera obtener una mejora con otro hardware se ejecutó el programa en una hoja de un Blade con 8 procesadores Intel(R) Xeon(R) CPUE5405 @ 2.00GHz, que tampoco mostró signos de mejora respecto al rendimiento, esta ejecución puede verse en la Figura 9.23.

Teniendo en cuenta lo anterior y gracias a que la versión paralela del programa

también está disponible en http://people.sc.fsu.edu/~jburkardt/f77_src/prime_openmp/prime_openmp.html, los resultados de dicha paralelización en lo que respecta a rendimiento no han sido excelentes. Pero, por el contrario un hecho destacable es que la paralelización realizada por los autores es exactamente igual a la realizada en forma automatizada por la transformación implementada. Ver código fuente en el Apendice B. Dicha paralelización resultó ser efectiva en un hardware de 16 núcleos con 8 threads.

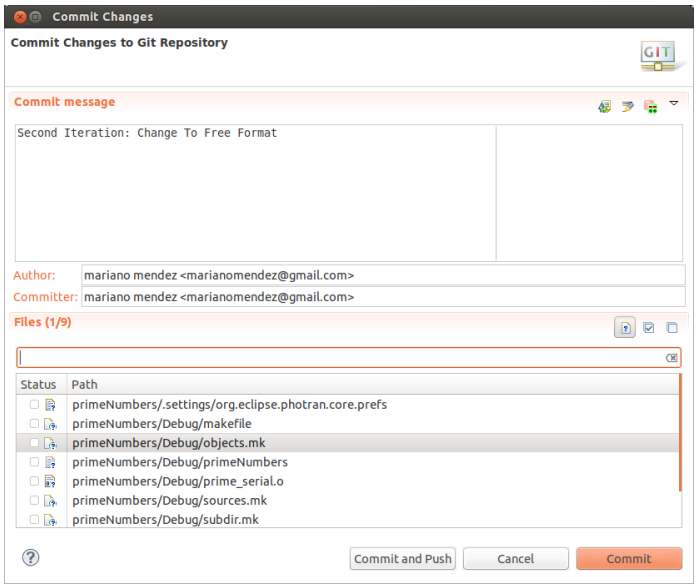


Figura 9.16: Establecimiento de la Versión de Trabajo, Bajo Control de Versiones en Git

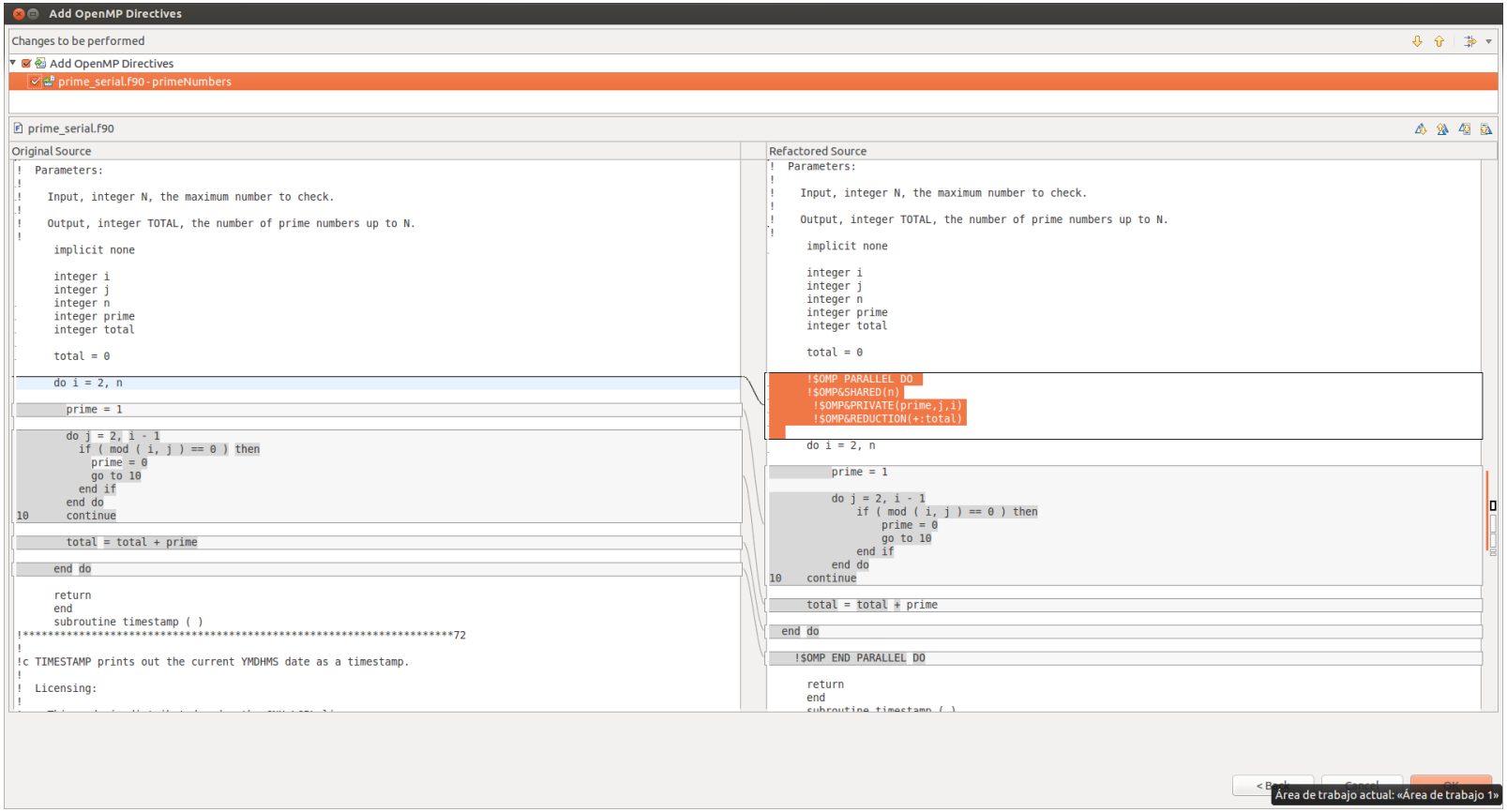


Figura 9.17: Vista de Diff Donde se Aprecia el Código Fuente Sin Transformar a la Izquierda de la Pantalla y el Código Fuente Transformado con las Directivas de OpenMP a la Derecha de la Misma

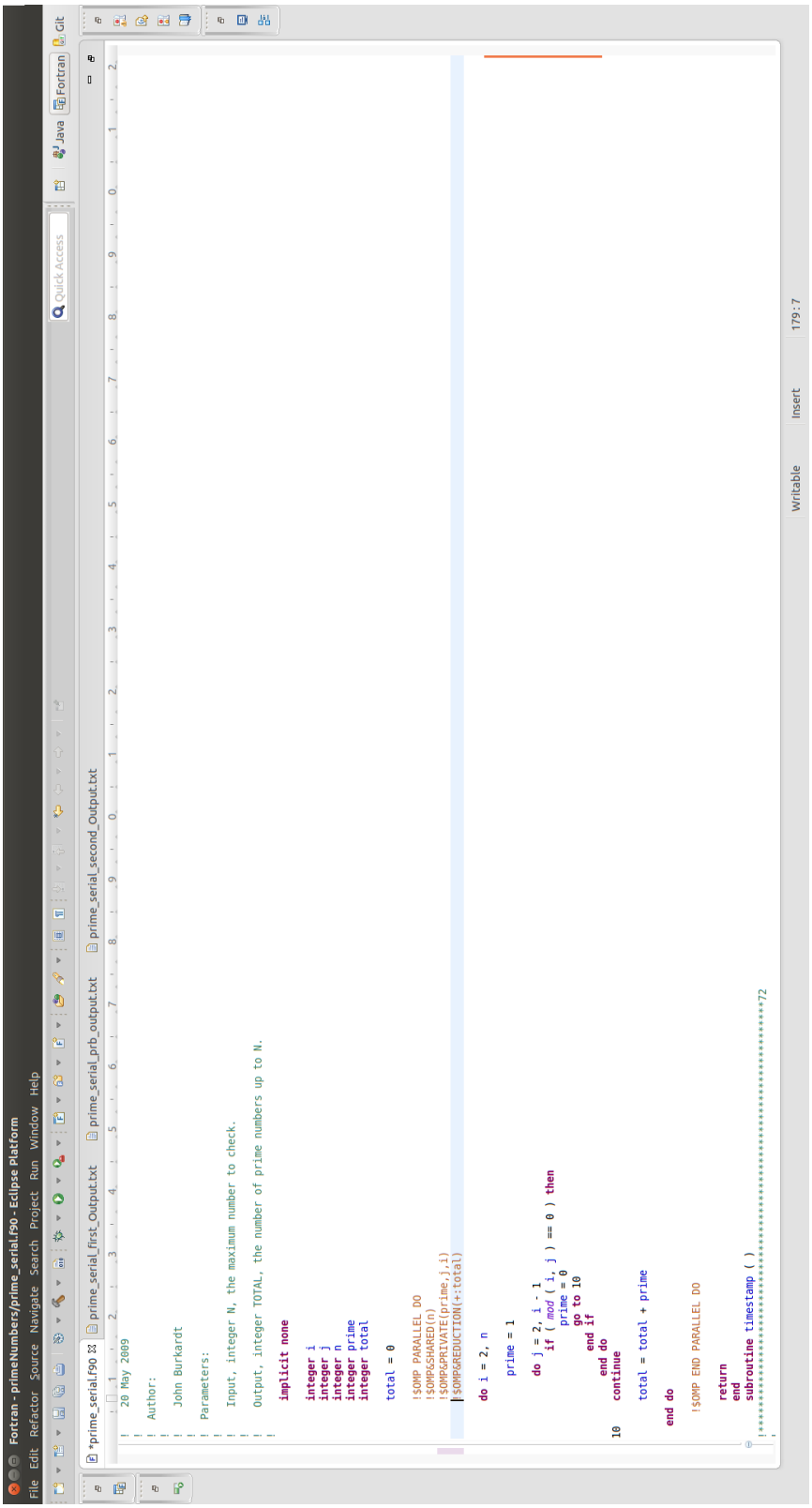


Figura 9.18: Código Fuente ya Transformado

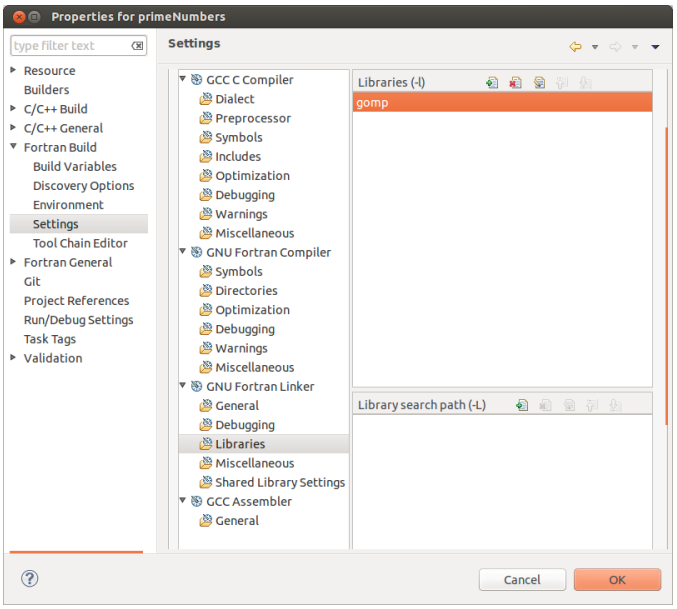


Figura 9.19: Establecimiento de la Versión de la Biblioteca de OpenMP

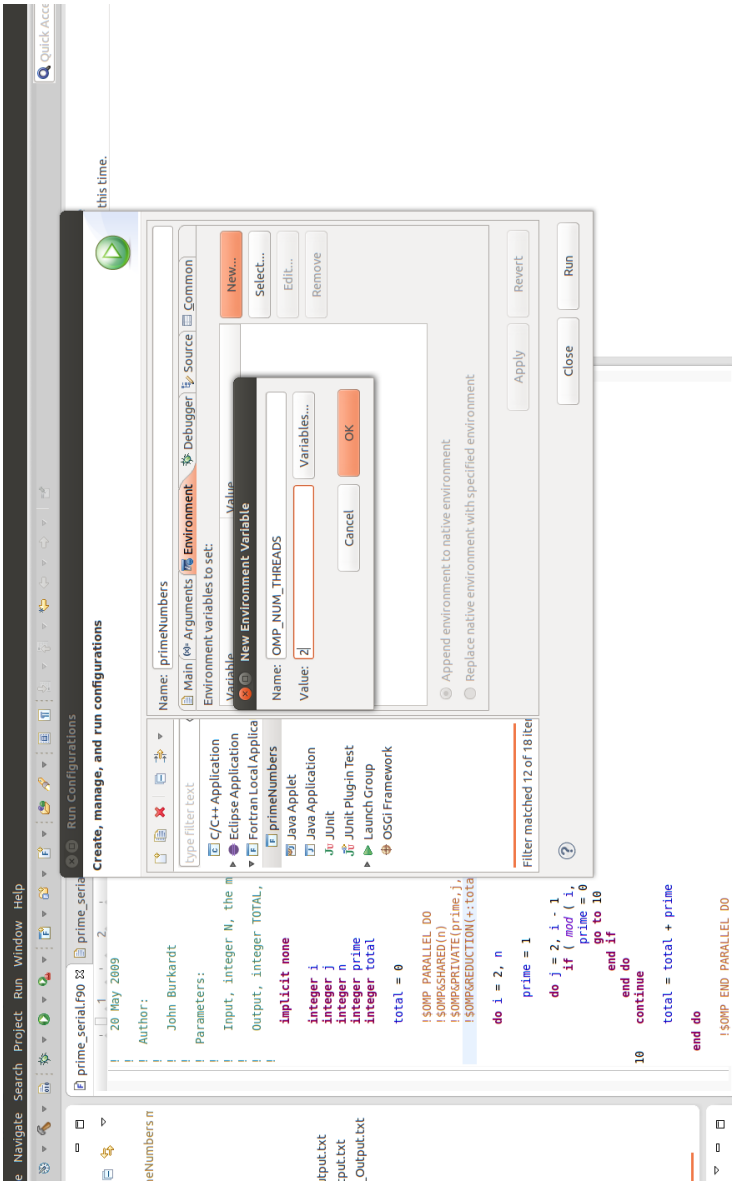


Figura 9.20: Opciones del Compilador

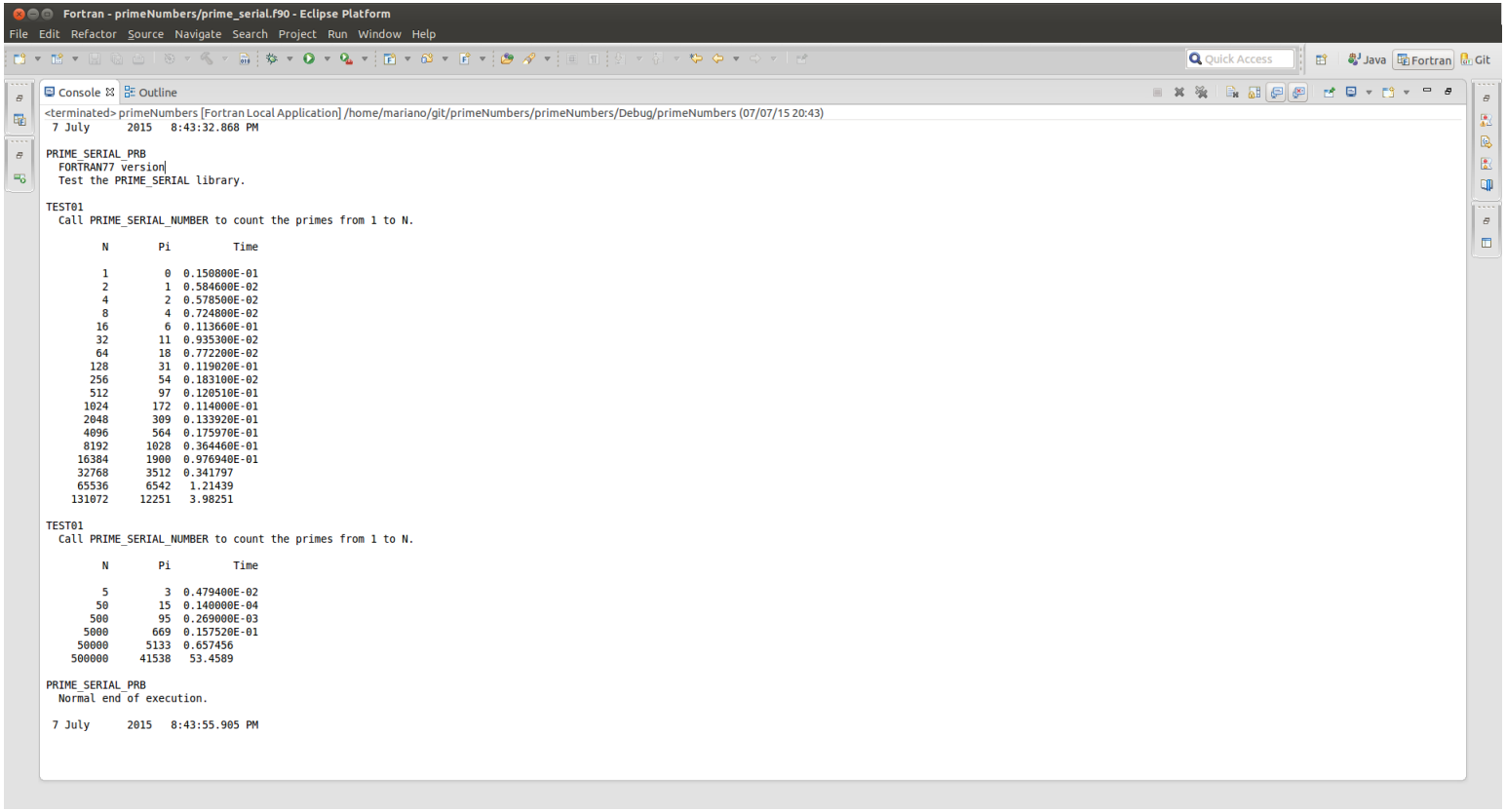


Figura 9.21: Ejecución del Programa en Paralelo

```
7 July      2015   8:43:32.868 PM

PRIME_SERIAL_PRB
FORTRAN77 version
Test the PRIME_SERIAL library.

TEST01
Call PRIME_SERIAL_NUMBER to count the primes from 1 to N.

N        Pi          Time

1         0  0.150800E-01
2         1  0.584600E-02
4         2  0.578500E-02
8         4  0.724800E-02
16        6  0.113660E-01
32       11  0.935300E-02
64       18  0.772200E-02
128      31  0.119020E-01
256      54  0.183100E-02
512      97  0.120510E-01
1024    172  0.114000E-01
2048    309  0.133920E-01
4096    564  0.175970E-01
8192   1028  0.364460E-01
16384  1900  0.976940E-01
32768  3512  0.341797
65536  6542   1.21439
131072 12251  3.98251

TEST01
Call PRIME_SERIAL_NUMBER to count the primes from 1 to N.

N        Pi          Time

5         3  0.479400E-02
50       15  0.140000E-04
500      95  0.269000E-03
5000    669  0.157520E-01
50000  5133  0.657456
500000 41538  53.4589

PRIME_SERIAL_PRB
Normal end of execution.

7 July      2015   8:43:55.905 PM
```

Figura 9.22: Resultados Ejecución en Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz 2 núcleos 4 Threads

```
7 July      2015  11:11:35.422 PM

PRIME_SERIAL_PRB
FORTRAN77 version
Test the PRIME_SERIAL library.

TEST01
Call PRIME_SERIAL_NUMBER to count the primes from 1 to N.

N        Pi          Time

1         0   0.00000
2         1   0.00000
4         2   0.00000
8         4   0.00000
16        6   0.00000
32       11   0.00000
64       18   0.00000
128      31   0.00000
256      54   0.00000
512      97   0.00000
1024    172   0.00000
2048    309   0.00000
4096    564   0.00000
8192   1028  0.280020E-01
16384  1900  0.136008
32768  3512  0.492030
65536  6542   1.68010
131072 12251   4.73230

TEST01
Call PRIME_SERIAL_NUMBER to count the primes from 1 to N.

N        Pi          Time

5         3   0.00000
50       15   0.00000
500      95   0.00000
5000    669  0.400000E-02
50000  5133   1.10407
500000 41538   51.0752

PRIME_SERIAL_PRB
Normal end of execution.
```

Figura 9.23: Resultados Ejecución en Hoja blade Intel(R) Xeon(R) CPUE5405 2.00GHz de 8 Procesadores

9.2. Caso de Estudio 4: Estimación del Valor de una Integral

El siguiente caso de estudio calcula la integral de la función utilizando un método que calcula promedios:

$$f(x) = \frac{50}{(pi * (2500 * x^2 + 1))}$$

para mayor información consultar la página del autor http://people.sc.fsu.edu/~jburkardt/f77_src/quad_serial/quad_serial.html. Al igual que en el caso anterior se planifica una iteración inicial, cuyo objetivo principal es tener al final de la misma el programa ejecutado con los resultados numéricos correctos y con un perfilado de la ejecución, ver Figura 9.24. En este caso la lista de cambios estaría conformado por un único cambio o transformación que consiste en tener el programa ejecutándose a partir de los fuentes.

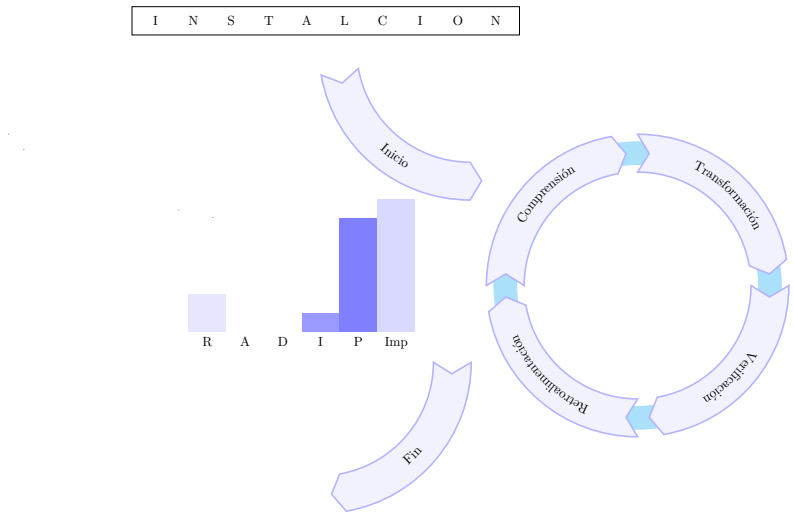


Figura 9.24: Proceso de Desarrollo Dirigido por el Cambio Primera Iteración caso de Estudio Integración Numérica

9.2.1. Iteración 1: Instalación y Perfilado

Al igual que en el caso anterior inicialmente se procede a realizar el primer cambio cuyo objetivo es tener el programa funcionando y el perfilado completo. Se seguirá el flujo de trabajo propuesto en el Capítulo 3, que consiste en:

1. Establecer una versión inicial del código fuente

2. Transformar el código fuente

3. Verificar el código fuente obtenido

4. Validar los resultados numéricos

5. Aceptar/Rechazar el cambio en base a los resultados numéricos

6. Documentar

El código fuente utilizado, considerado legacy, que cuadra dentro de varias de las características de las definiciones de código legacy, se descarga de `http://people.sc.fsu.edu/~jburkardt/f77_src/quad_serial/quad_serial.html`, se utiliza Eclipse y el plug-in Photran 8 como herramienta de desarrollo, PhotranLint plug-in para Photran desarrollado en este trabajo, el compilador que se utiliza es gfortran versión 4.8. Como software de control de versionado se utiliza git y su plug-in para eclipse, al igual que en el ejemplo anterior. En esta primera iteración se crea un proyecto Fortran dentro del IDE, a continuación se descargan los fuentes del programa y se agregan dentro del proyecto, seguidamente se setean las correspondientes opciones del compilador y del linkeditor, al igual que en el caso de estudio anterior. Luego se pone bajo control de versiones el código fuente obtenido y la salida original del programa, ver Figura 9.29.

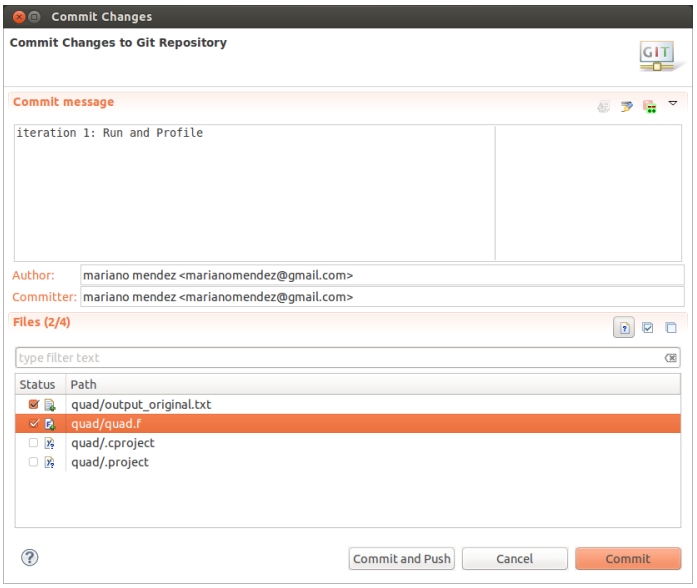


Figura 9.25: Vista del Plug-in Egit de Eclipse con el Primer Envío de Código Fuente

Una vez que todos estos pasos se han llevado a cabo se procede a la compilación del programa por primera vez, ver Figura 9.26.

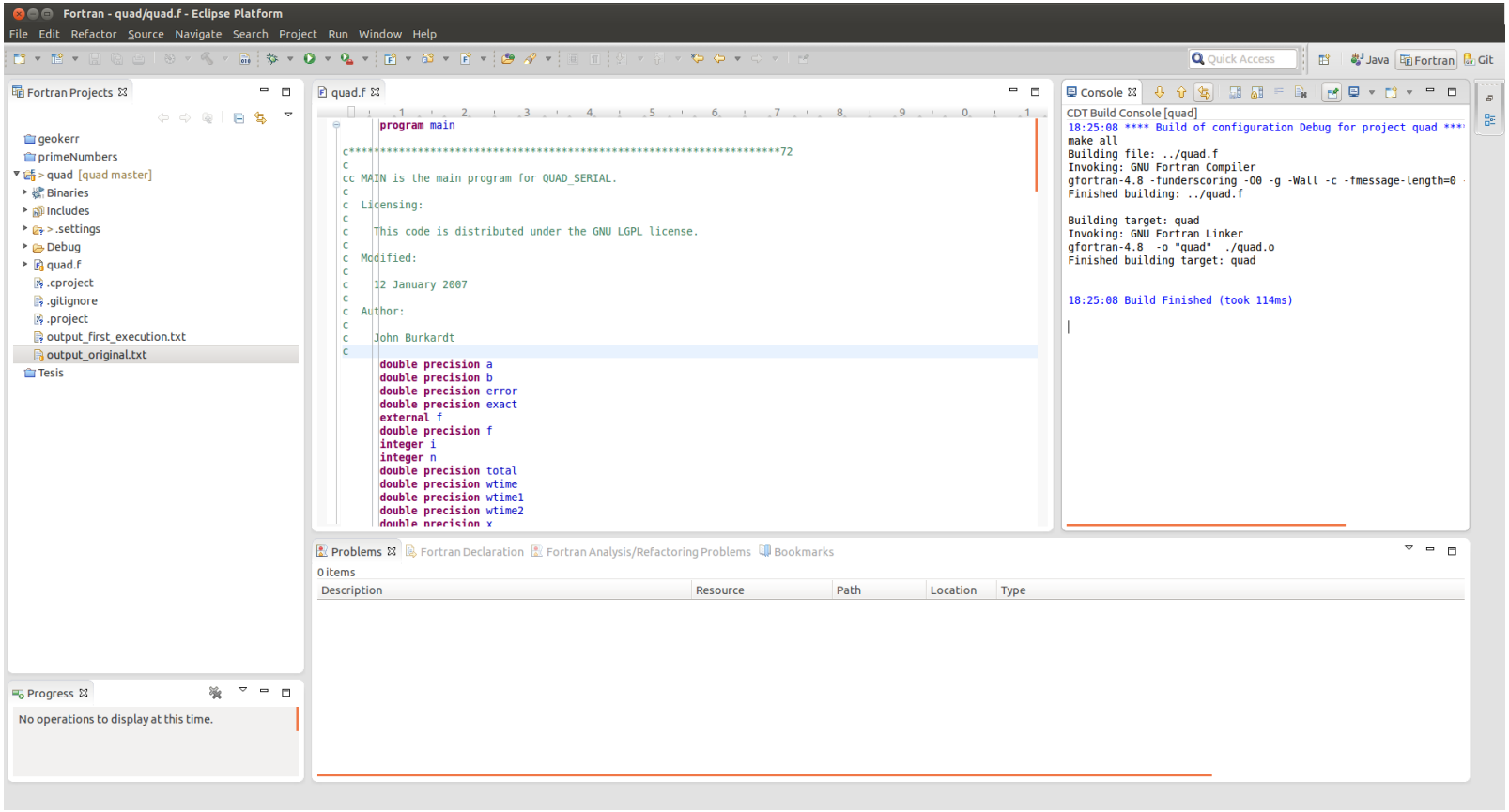


Figura 9.26: Compilación del Programa Quad.f

Cuando el programa se ha compilado correctamente, acción que a veces no es trivial, se procede a la ejecución del mismo para la obtención de los resultados. Este ejemplo ha sido seleccionado para poder contrastar los datos obtenidos con datos que hayan sido obtenidos por los autores del mismo. A continuación puede verse que la ejecución inicial es exactamente igual que la obtenida:

Ejecución Original	Ejecución Inicial
14 December 2011 8:28:12.640 AM	16 July 2015 6:25:48.719 PM
QUAD:	QUAD:
FORTRAN77 version	FORTRAN77 version
Estimate the integral of f(x) from A to B.	Estimate the integral of f(x) from A to B.
f(x) = 50 / (pi * (2500 * x * x + 1)).	f(x) = 50 / (pi * (2500 * x * x + 1)).
A = 0.00000	A = 0.00000
B = 10.0000	B = 10.0000
N = 10000000	N = 10000000
Exact = 0.499363	Exact = 0.499363
Estimate = 0.499371	Estimate = 0.499371
Error = 0.790784E-05	Error = 0.790784E-05
Time = 0.338820	Time = 0.308151
QUAD_SERIAL:	QUAD_SERIAL:
Normal end of execution.	Normal end of execution.
14 December 2011 8:28:12.980 AM	16 July 2015 6:25:49.027 PM

Esto puede comprobarse en la Figura 9.27 que muestra la ejecución del programa una vez compilado. Y posteriormente, en la Figura 9.28 que muestra la vista de diferencia o Diff entre las dos corridas, en la cual las fechas y el valor del tiempo de ejecución del programa son los únicos valores que cambian entre las ejecuciones.

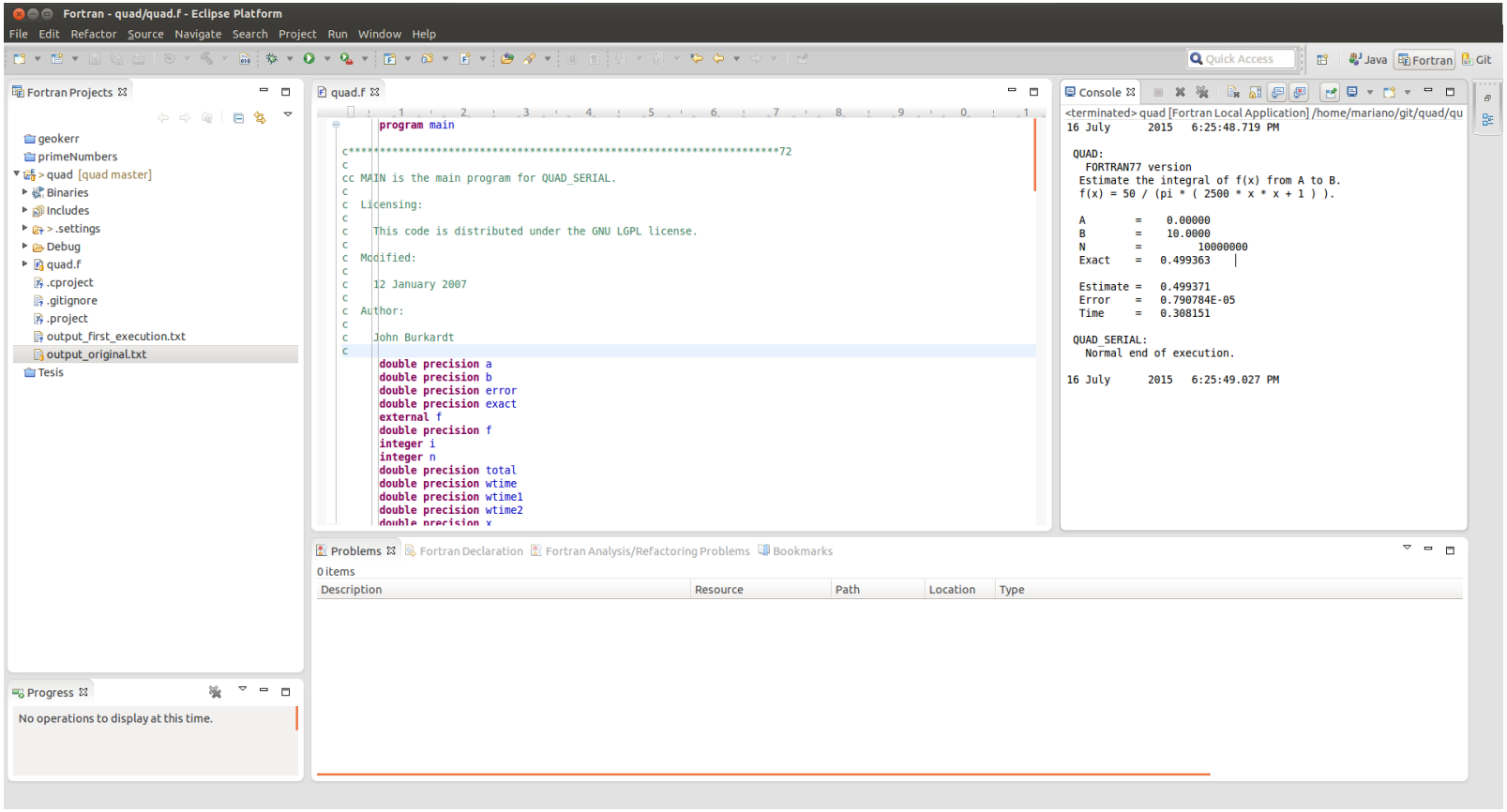


Figura 9.27: Resultados de la Ejecución del Programa

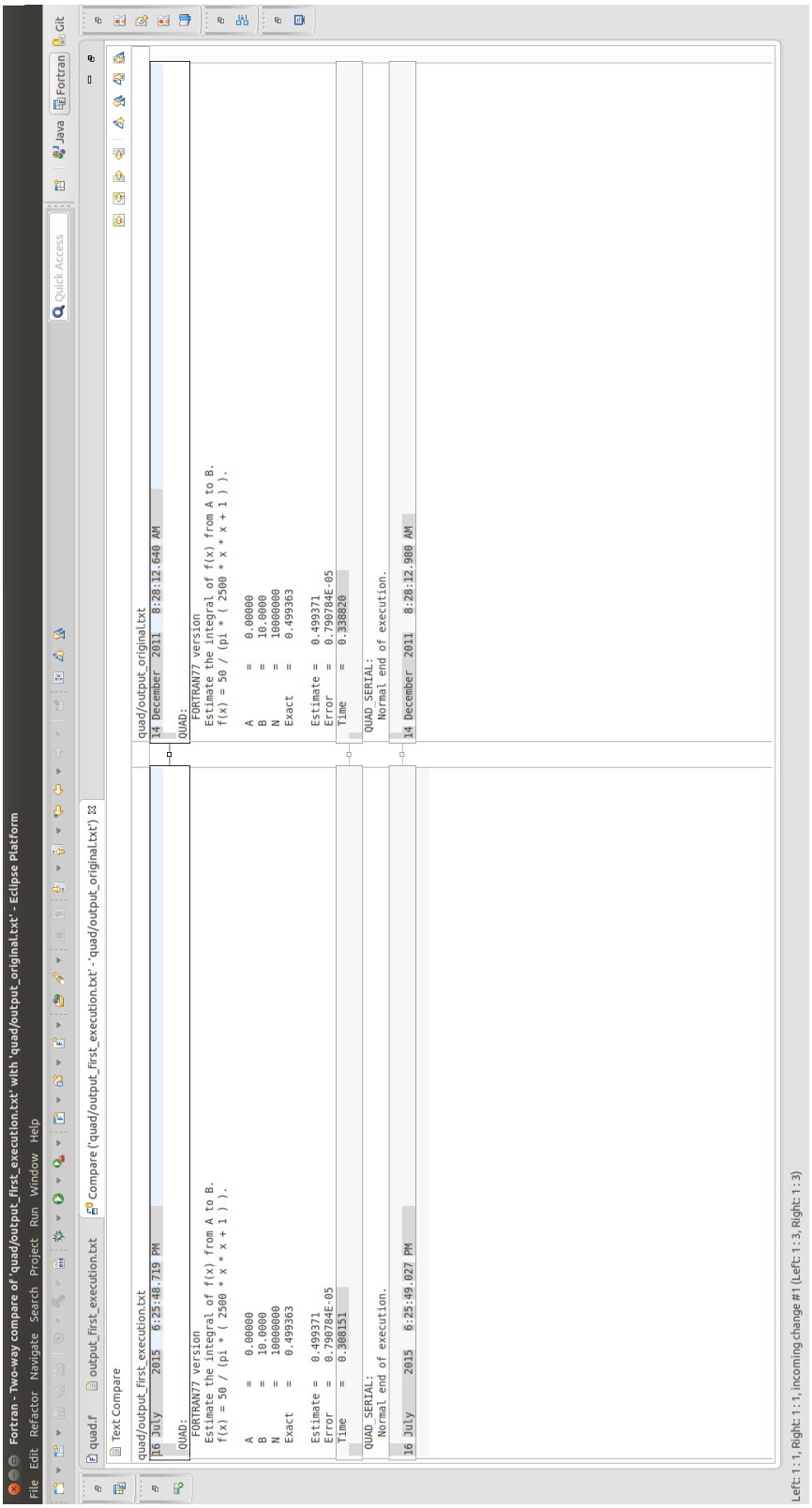


Figura 9.28: Vista de Diff entre la Ejecución Original y la Ejecución Obtenida en el IDE

De la etapa de retroalimentación del ciclo de Desarrollo Guiado por el Cambio (CDD), se desprenden dos iteraciones nuevas:

- Cambiar a formato libre
- Paralelizar la instrucción DO de la línea 61

La iteración en la que se propone paralelizar surge del análisis del perfilado de la ejecución del programa:

```
ndex % time    self  children    called     name
<spontaneous>
[1]     92.3    0.07    0.17                 MAIN__ [1]
0.17    0.00 10000000/10000000     timestamp_ [2]
0.00    0.00       1/2           register_tm_clones [5]
-----------------------------------------------
0.17    0.00 10000000/10000000     MAIN__ [1]
[2]     65.4    0.17    0.00 10000000         timestamp_ [2]
-----------------------------------------------
0.02    0.00       1/1           main [4]
[3]      7.7    0.02    0.00       1         f_ [3]
0.00    0.00       1/2           register_tm_clones [5]
-----------------------------------------------
<spontaneous>
[4]      7.7    0.00    0.02                 main [4]
0.02    0.00       1/1           f_ [3]
-----------------------------------------------
0.00    0.00       1/2           f_ [3]
0.00    0.00       1/2           MAIN__ [1]
[5]      0.0    0.00    0.00       2         register_tm_clones [5]
-----------------------------------------------
```

En los resultados del perfilado la función MAIN() es aquella en la que se realiza el mayor porcentaje del cómputo. Observando el código fuente la instrucción DO es la candidata ideal para la paralelización del mismo.

9.2.2. Iteración 2: Cambiar a formato libre

Al igual que en los casos de estudio anteriores, esta transformación ha resultado ser de suma utilidad. Se procede a poner bajo versionado de código la versión actual, ver Figura 9.29.

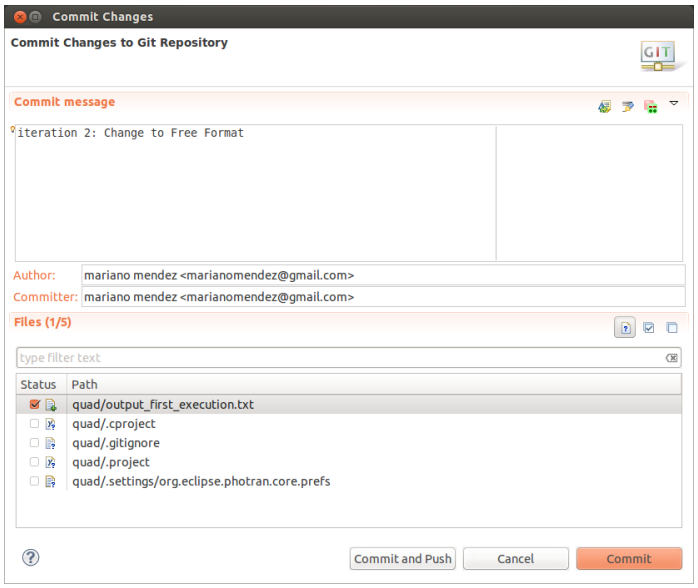


Figura 9.29: Vista del Plug-in Egit de Eclipse con el Segundo Envío de Código Fuente

Una vez realizado el commit se procede a llevar a cabo la transformación ver Figura 9.30, posteriormente se compila y ejecuta el código fuente transformado, ver Figura 9.31. Por último, se realiza la comparación de los resultados numéricos, ver Figura 9.32.

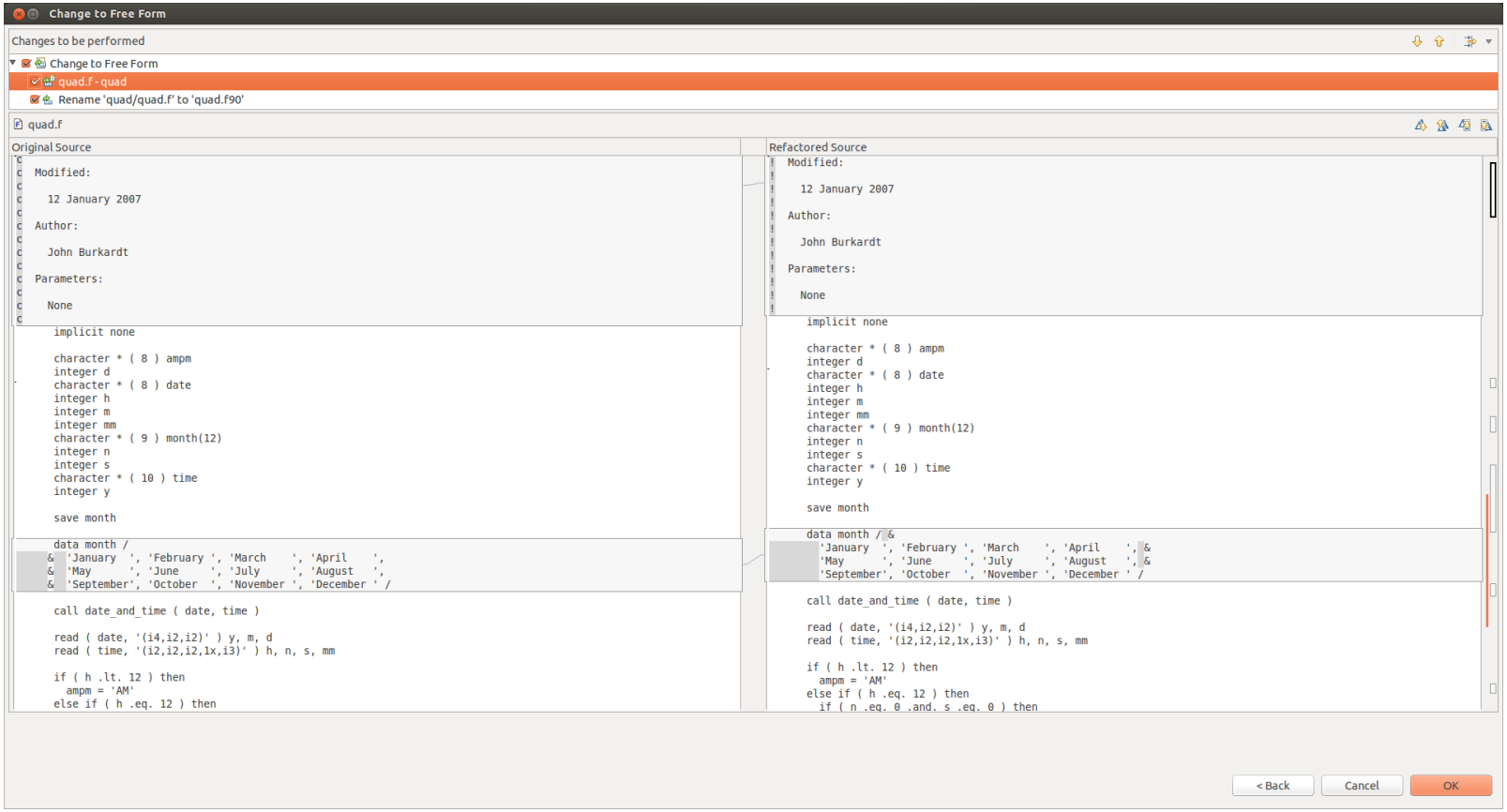


Figura 9.30: Vista Diff del Código Fuente Previo a la Transformación a la Derecha, y Posterior al Cambio a la Izquierda

Figura 9.31: Resultados de la Ejecución del Programa

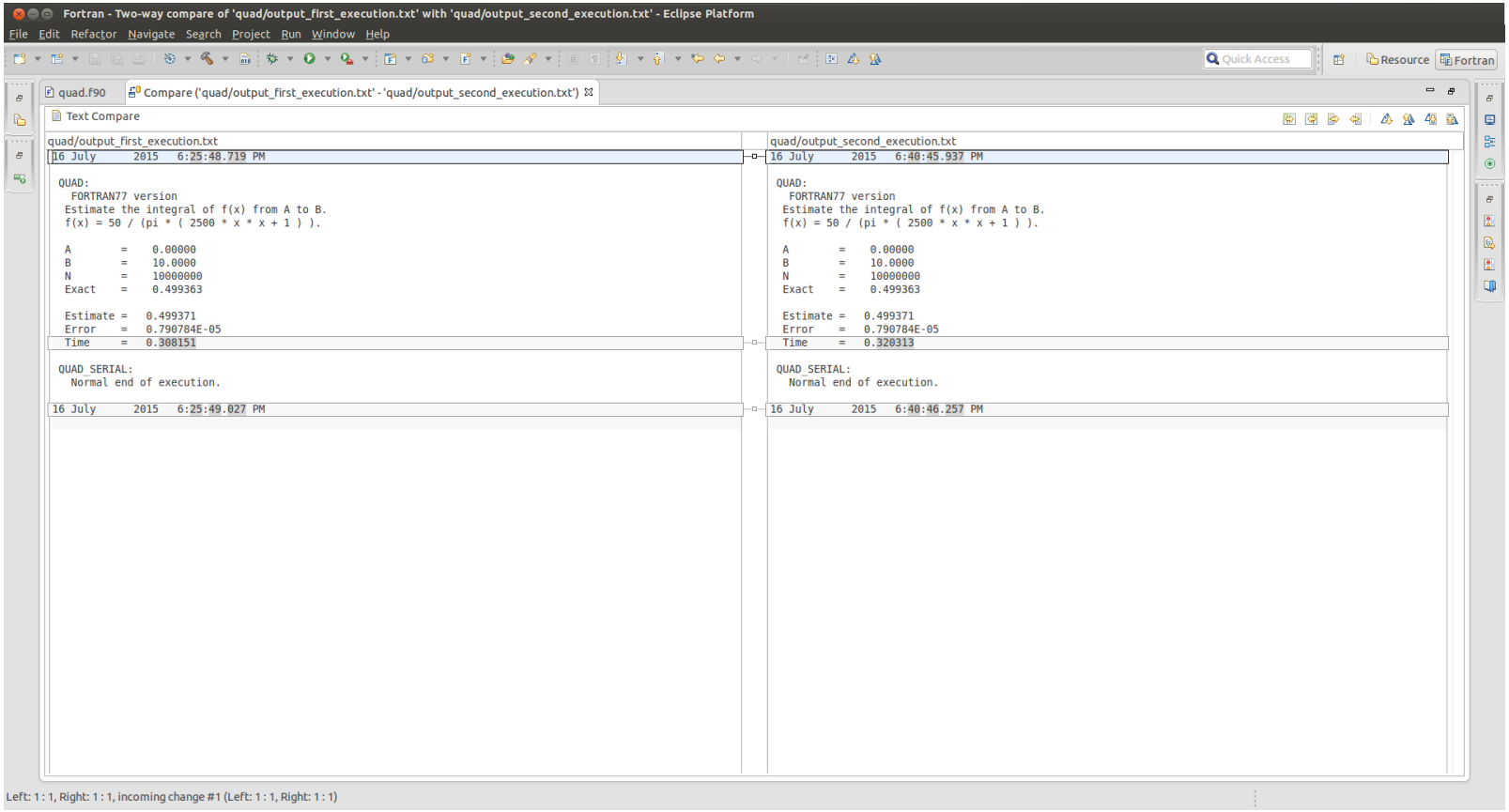


Figura 9.32: Vista de Diff Entre la Ejecución Original y la Ejecución Obtenida en el IDE

9.2.3. Iteración 3: Paralelización

Como en todas las iteraciones se comienza con fijar una versión del código fuente como versión inicial de dicha iteración. Esta versión siempre debe garantizar que los resultados numéricos sean los mismos que aquellos de la versión anterior. Se utilizará Egit para fijar la versión previa a la transformación de la iteración 3, ver Figura 9.34. A continuación en la Figura 9.33 se puede apreciar los efectos de la transformación sobre el código fuente en cuestión.

Code Before

```
total = 0.0D+00


do i = 1, n
x = ( ( n - i ) * a + ( i - 1 ) &
 * b ) / ( n - 1 )
total = total + f ( x )
end do
```

Code After

```
total = 0.0D+00

!$OMP PARALLEL DO  &
!$OMP SHARED(b,a,n) &
!$OMP PRIVATE(x,i) &
!$OMP REDUCTION(+:total)

do i = 1, n
x = ( ( n - i ) * a + ( i - 1 ) &
 * b ) / ( n - 1 )
total = total + f ( x )
end do
!$OMP END PARALLEL DO
```

Figura 9.33: Trandofrmación Automática de Código Fuente para Paralelización

Una vez establecida la versión de trabajo y tras haber analizado en la iteración 1 que el único foco de cómputo es la instrucción DO de la línea 61, que se lista a continuación:

```
do i = 1, n
    x = ( ( n - i ) * a + ( i - 1 ) * b ) / ( n - 1 )
    total = total + f ( x )
end
```

Se procede a aplicar la transformación de código fuente implementada como un refactoring, para determinar si el mismo es paralelizable. Se realizará una serie de verificaciones y de resultar paralelizable, la misma transformación mostrará el resultado de la paralelización, ver Figura 9.35, Figura 9.36 y Figura 9.37. Es necesario configurar el IDE para que la compilación incluya las bibliotecas de OpenMp,

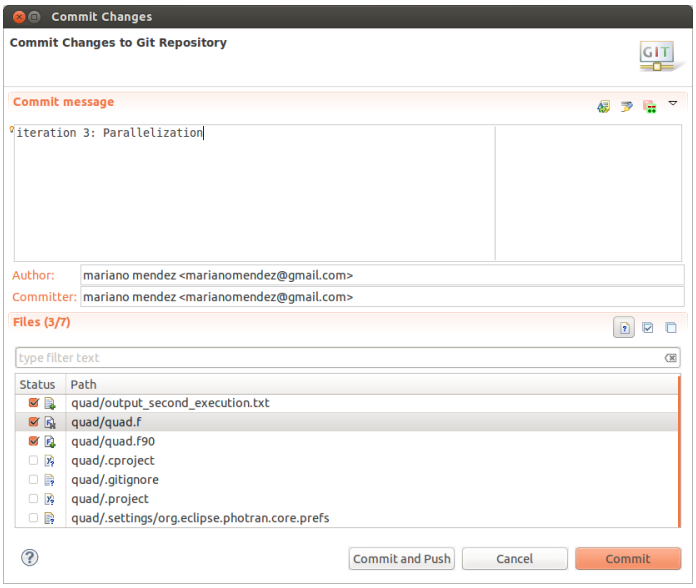


Figura 9.34: Commit de la Versión en el Repositorio Previa a la Paralelización

para ello se agregarán las opciones de compilación y linkedición. Esto puede ser observado en la Figura 9.38, Figura 9.39, Figura 9.40 y la Figura 9.41. Posteriormente se procede a la compilación y ejecución del programa cuyos resultados pueden verse en la Figura 9.42 en la cual se puede apreciar la mejora en el tiempo de ejecución del programa que pasa de 0.320313 segundos a 0.102887 segundos, obteniendo una mejora de aproximadamente el 33 % manteniendo los mismos resultados numéricos. Los resultados de las ejecuciones del programa pueden verse en el Apéndice B.

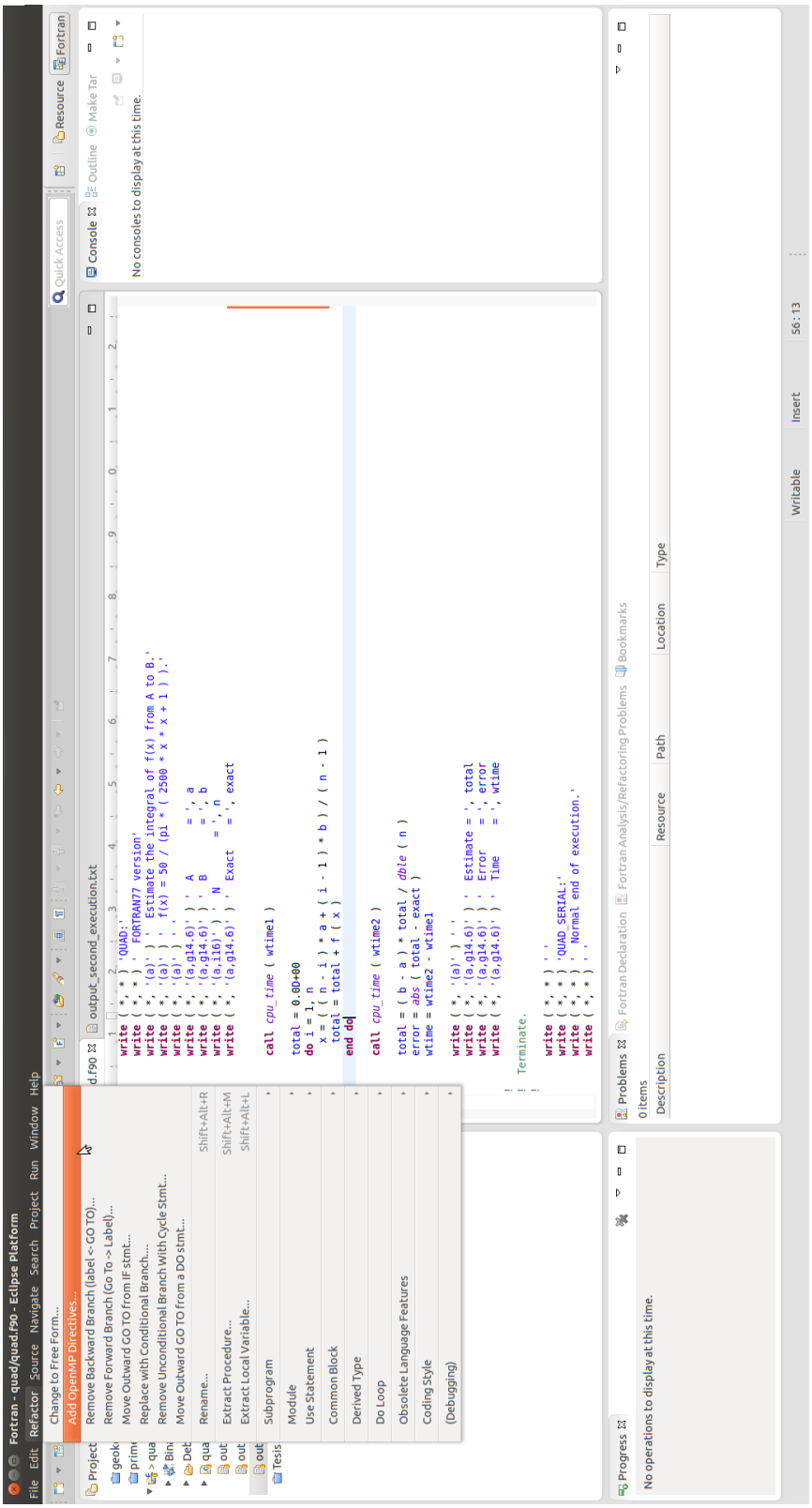


Figura 9.35: Selección de la Opción de Menú de la Transformación para Paralelizar

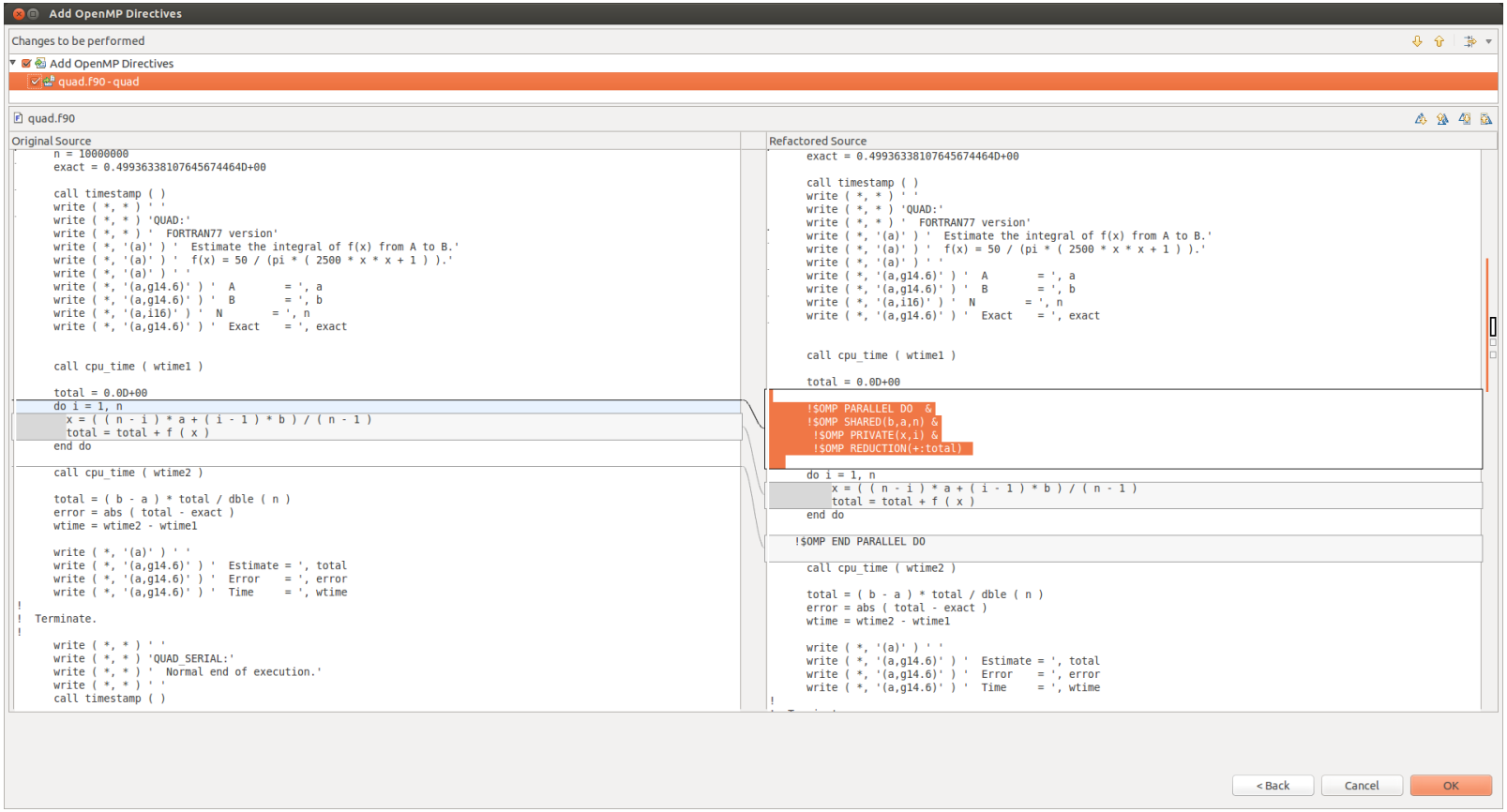


Figura 9.36: Vista de Diff entre la Versiòn del Código Fuente Sin Paralelizar y el Paralelizado

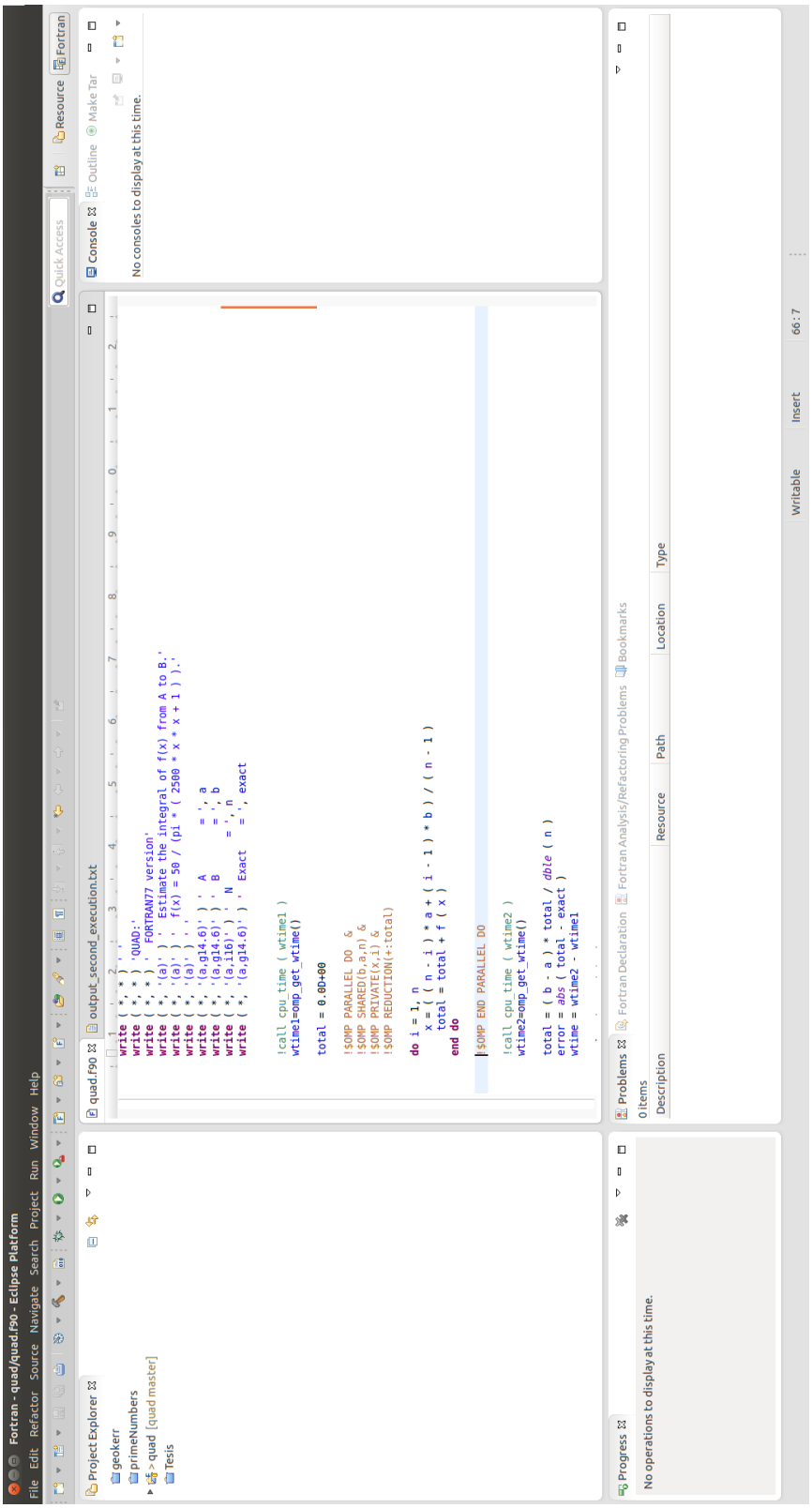


Figura 9.37: Editor del IDE con el Código Fuente Paralelizado

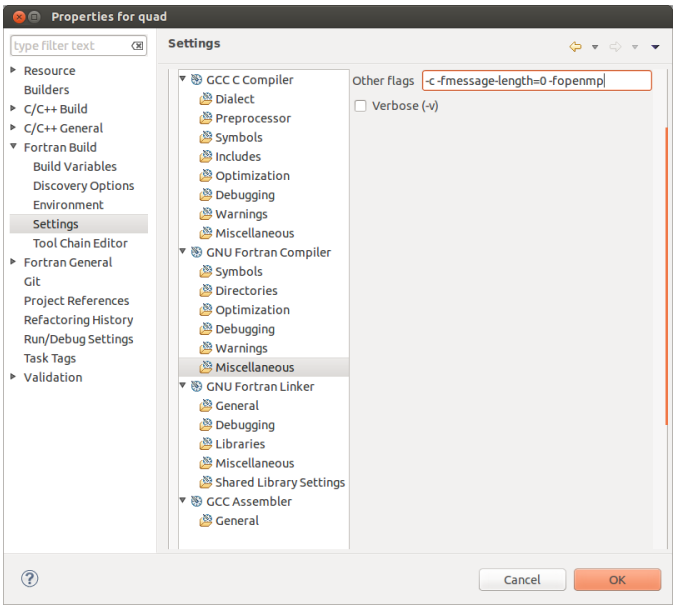


Figura 9.38: Congiguracion de Parámetros de Compilación

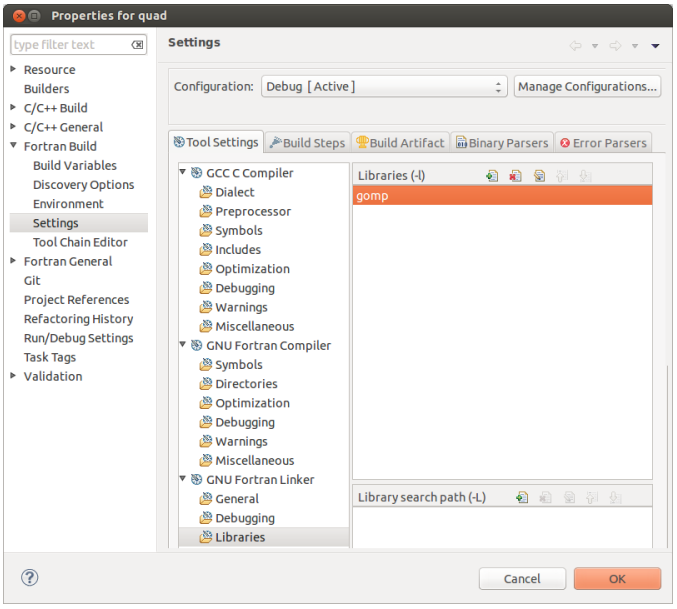


Figura 9.39: Configuración de Parámetros de Linkedición

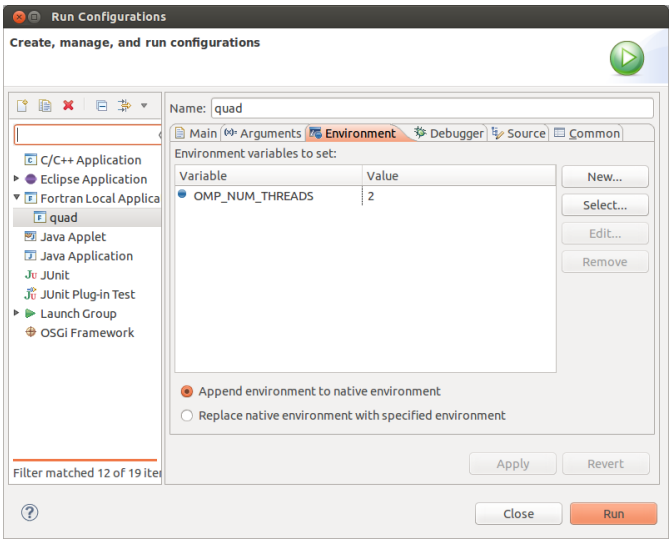


Figura 9.40: Seteo del Números de Threads de OpenMP

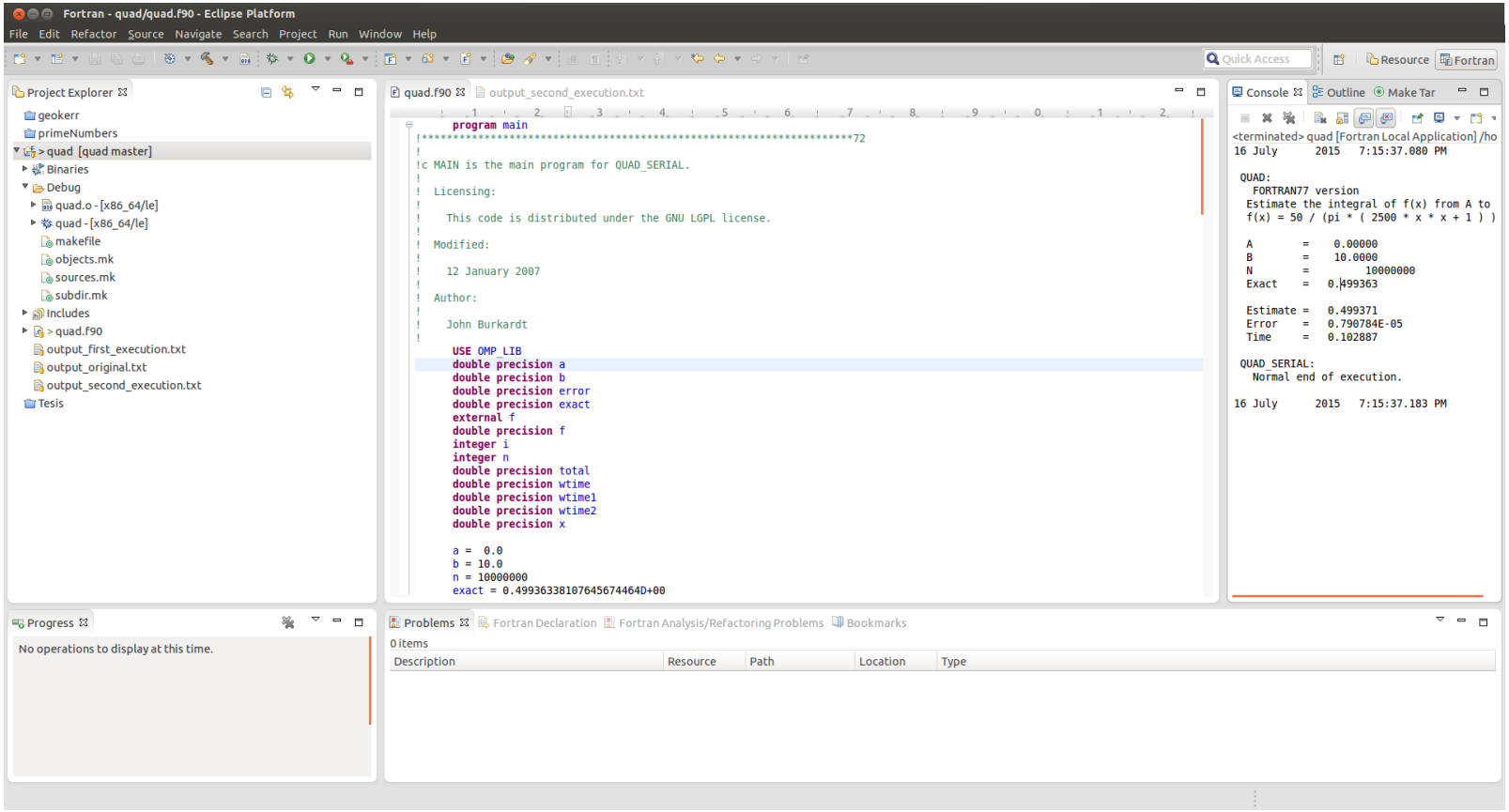


Figura 9.41: Ejecución y Resultados del Programa ya Paralelizado

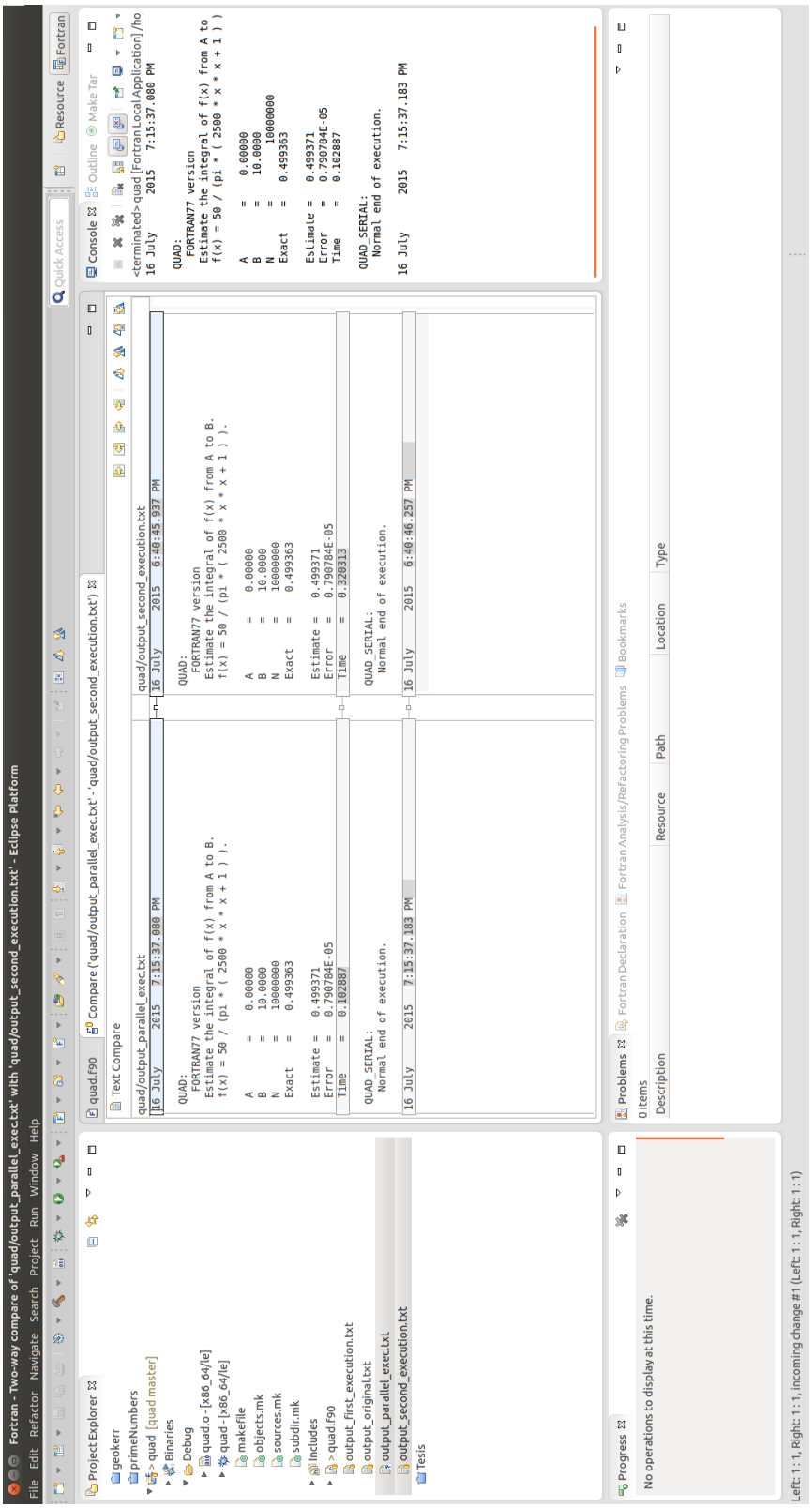


Figura 9.42: Vista de Diff entre la Ejecución Original y la Ejecución Paralela Obtenida en el IDE

9.3. Resumen

En este capítulo se ha aplicado el proceso de desarrollo dirigido por el cambio mediante la ejecución del flujo de trabajo propuesto en cada una de las iteraciones planteadas con el objetivo de actualizar dos programas escritos inicialmente en FORTRAN 77 a una versión más moderna del lenguaje como es Fortran 90, con el agregado de haber transformado el código fuente secuencial en código fuente que puede ser ejecutado en forma paralela en un entorno de multicore con memoria compartida, sin haber alterado la funcionalidad del programa. Esta transformación ha sido llevada a cabo en forma completamente automatizada. Además los resultados obtenidos tras la ejecución del código fuente paralelizado muestran mejoras significativas en el rendimiento de la ejecución de uno de los programas.

Capítulo 10

Conclusiones

En este capítulo proporcionamos algunas observaciones finales, se retoman los aportes de esta tesis y se describen las futuras líneas de investigación así como los posibles trabajos futuros a ser encarados.

10.1. Contribuciones

En este trabajo se ha abordado el problema de la actualización de código fuente de aplicaciones científicas heredadas a través de la definición, descripción e implementación de un nuevo proceso de desarrollo. Este proceso de desarrollo posee tres características que lo distinguen:

- Es un ***proceso iterativo*** e ***incremental***.
- Está ***dirigido por el cambio o la transformación***, una de las cuatro características esenciales del software.
- Está ***centrado en las herramientas de desarrollo***.

Teniendo en cuenta estas características que definen al proceso se lo ha denominado "***Desarrollo Guiado por el Cambio***" (**Change Diven Development**). Si bien fue pensado inicialmente para la etapa de mantenimiento del software, se determinó que también podría ser utilizado para la construcción de software desde

cero. Además se ha propuesto y establecido un flujo de trabajo para llevar a cabo en cada una de las iteraciones definidas el proceso, ver figura 10.1 en la cual se muestra dicho proceso y la composición de un cambio en las actividades centrales del desarrollo de software (Requerimientos, Análisis, Diseño, Implementación, Pruebas e Implantación).

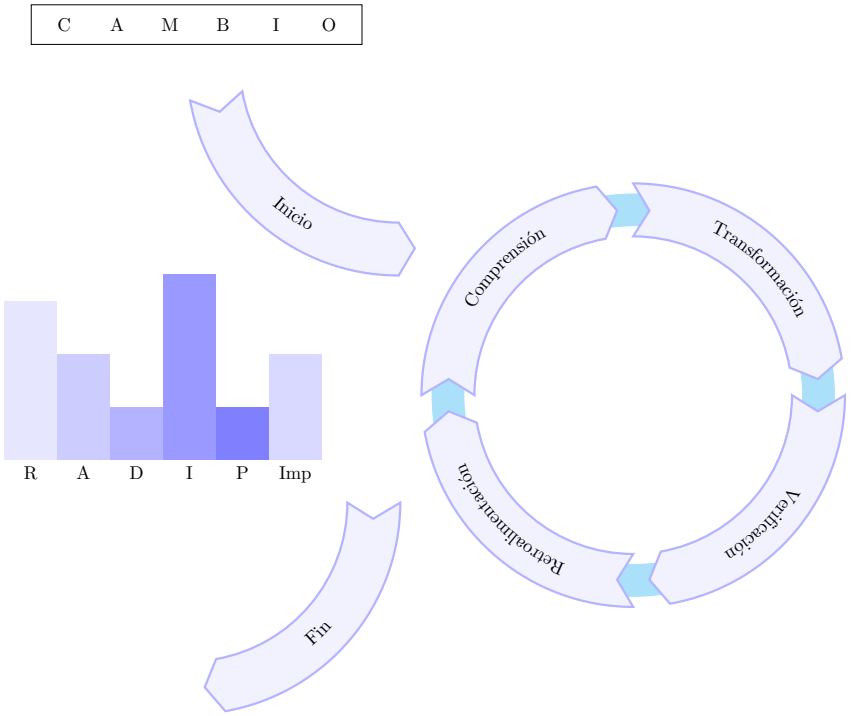


Figura 10.1: Un Ciclo Completo del Proceso de "Desarollo Guiado por el Cambio" (Change Diven Development)

Asimismo se ha descrito detalladamente cada una de las etapas del proceso propuesto y de las herramientas que han sido implementadas para ser utilizadas en cada una de ellas. Dichas herramientas destinadas a facilitar el proceso de aplicación de cambios de código fuente Fortran de aplicaciones de cómputo científico heredadas se han integrado a un entorno de desarrollo llamado Eclipse. El conjunto de estas herramientas construido a lo largo de este trabajo de investigación ha sido agrupado en un Plug-in de Eclipse llamado ***PhotranLint***. El mismo reúne, en un único producto, cada una de las herramientas presentadas e implementadas en este trabajo:

- **Herramientas de Comprensión**: Análisis Estático de COMMON BLOCKS, Árbol Estático de Llamadas, etc.
- **Herramientas de Transformación**: Refactorizaciones de Photran para Código Fortran.
- **Herramientas de Medición**: Contadores de Hardware, Métricas, etc.

Destacando la implementación en forma de refactorizaciones para Photran de las transformaciones automáticas de código fuente, el proceso ha sido aplicado a varios ejemplos y otros documentados en esta tesis de manera completa en los capítulos 7, 8 y 9. Además se han propuesto algunas transformaciones de código fuente Fortran que se agregan al catálogo descrito en [149, 148]. Esta tesis propone encarar el desarrollo de software centrándose en una de las propiedades esenciales del mismo: ***El Cambio***.

El enfoque innovador presentado en este trabajo se basa en la utilización de transformaciones automáticas de código fuente Fortran (el cambio) integradas a un IDE con el formato de refactorizaciones (las herramientas) como la piedra fundamental (el proceso) del desarrollo de aplicaciones científicas heredadas ya sea desde cero o en su etapa de mantenimiento. Basados en la idea de que desarrollar software no es otra cosa que transformar (cambiar) programas [119].

Durante el desarrollo de este trabajo de investigación se han publicado, implementado, integrado y mostrado resultados obtenidos durante el proceso. En "**First Steps Towards a Tool for Legacy Systems**" [154] se destaca la necesidad de tener un proceso definido para la etapa de mantenimiento del software heredado escrito en Fortran. Además se pone en evidencia la necesidad de satisfacer la existencia de una herramienta que implemente, integre y provea los requerimientos de los desarrolladores que se ocupan de las tareas de mantenimiento del software heredado en Fortran. En el artículo "**Restructuring Fortran Legacy Applications for Parallel Computing in Multiprocessors**" [203] se proponen e implementan algunas transformaciones automáticas para paralelizar código fuente secuencial escrito en Fortran. Además se implementan algunos análisis para

determinar si es posible paralelizar algunas partes del código fuente Fortran secuencial. En "**Aplicaciones Científicas Numéricas: El (Ciclo de Vida del) Software Heredado**" [155] se propuso la línea de investigación en la cual se encuadraría el siguiente trabajo. A su vez en "**Fortran Legacy Software: Source Code Update and Possible Parallelization Issues**" [201] se proponen e implementan un conjunto de transformaciones de código fuente escrito en Fortran tendientes a mejorar la legibilidad y comprensibilidad del mismo como paso previo a la paralelización. Volviendo a destacar la necesidad de un proceso de desarrollo para el Software Legacy así como también la necesidad de mejora e integración de herramientas de análisis y transformación de código fuente. En "**Legacy Fortran Software: Applying Syntactic Metrics to Global Climate Models**" [152] se implementó un conjunto de métricas para medir y comprender cómo estaban escritos ciertos programas del área del modelado del clima, escritos en Fortran. En este trabajo se pudo comprobar, a través de las métricas, que dichos programas utilizaban todavía varias de las características obsoletas de Fortran. Además en "**Optimization and Parallelization Experiences Using Hardware Performance Counters**" [200] se utilizaron los contadores de hardware para medir optimizaciones hechas en el código fuente, resultó que no siempre la información dada por los mismo arrojaba datos útiles para el proceso de paralelización. También se implementó una herramienta capaz de introducir y eliminar automáticamente mediciones a través de contadores de hardware en cualquier parte del código fuente seleccionada por un programador. Esta implementación fue presentada en "**An Automated Approach to Hardware Performance Monitoring Counters**" [202]. A su vez se construyeron una serie de métricas de software más elaboradas que las implementadas en [152]. Estas métricas se centran en la complejidad del software y fueron integradas a Photran un Plugin de Eclipse. La implementación permitió observar los valores de las mismas mientras el código fuente era editado. Por último, en "**Climate Models: Challenges for Fortran Development Tools**" [153] se estudió un conjunto bien definido de aplicaciones de modelado del clima, intervinientes activamente en el estudio del cambio climático, todas ellas escritas en Fortran. De dicho estudio se obtuvieron datos tendientes a determinar

cuales son las necesidades y retos que tendrá que tener una herramienta moderna dedicada al desarrollo de software para el lenguaje de programación Fortran.

Finalmente se ha intentado desmitificar la creencia que rige en ciertos ámbitos académicos de la Ciencia de la Computación en el cual se piensa a Fortran como un lenguaje arcaico y a sus programadores como ajenos las prácticas modernas de la ingeniería de software.

10.2. Trabajos Futuros

A partir de la elaboración de esta tesis, se abre un abanico de posibles líneas de investigación asociadas que no fueron consideradas en el desarrollo de la tesis pero cuya importancia ameritan ser tenidas en cuenta en futuros trabajos. Detallaremos, a continuación, los temas que no hemos considerados y cuya solución implica en si misma una línea de investigación a desarrollar:

1. Mejorar y consolidar el proceso de Desarrollo Dirigido por el Cambio (Change Driven Development).

2. Determinar distintos flujos de trabajos para aplicar al Desarrollo Dirigido por el Cambio (Change Driven Development).

3. Trasladar y aplicar a la industria del software en general los conceptos y técnicas utilizadas en el Desarrollo Dirigido por el Cambio (Change Driven Development) y en este trabajo de investigación.

4. Determinar, definir e implementar más transformaciones automáticas para Fortran.

5. Estudiar el caso de COBOL desde el punto de vista de su evolución y del Desarrollo Dirigido por el Cambio (Change Driven Development).

6. Estudiar transformaciones automáticas destinadas exclusivamente a la paralelización de código fuente como las propuestas en [203, 201].

7. Aplicar el proceso de Desarrollo Dirigido por el Cambio (Change Driven Development) para la paralelización de código fuente escrito en Fortran.

8. Estudiar transformaciones específicas para COMMON Blocks

9. Estudiar transformaciones específicas para la eliminación de código no estructurado.

10. Estudiar las necesidades de distintas ramas de la ciencia que utilizan a Fortran como herramienta de trabajo para determinar que tipo de herramientas son necesarias.

11. Estudiar la aplicación de conjuntos de transformaciones y si existe un patrón de uso con algunas de ellas.

12. Hacer un estudio detallado y multidisciplinar de como es utilizado el lenguaje de programación Fortran en la actualizada por los científicos.

13. Implementar más herramientas e integrarlas a PhotranLint.

14. Identificar automáticamente puntos susceptibles a ser cambiados, mejorados o actualizados teniendo en cuenta distintos criterios configurables, por ejemplo “Bad Smells”.

Apéndice A

Código Fuente y Resultados del Caso de Estudio 1

A.1. Programa: FIRST.f

Código Fuente del programa FIRST.F a través de las iteraciones:

A.1.1. Versión Original

```
   PROGRAM FIRST: FIRST.f
C NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
C
C FILE: first.f
C
C FIRST PROGRAMMING EXPERIMENT
C
   DATA  N/25/, H/1.0/, X/0.5/
   F = SIN(X)
   G = COS(X)
   DO 2 I = 1,N
   H = 0.25*H
   D = SIN(X + H) - F
   Q = D/H
   E = ABS(G - Q)
   PRINT *,H,D,Q,E
2  CONTINUE
   STOP
   END
```

A.1.2. Versión Iteración 1

```
PROGRAM FIRST
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
    DATA  N/25/, H/1.0/, X/0.5/
    F = SIN(X)
    G = COS(X)
    DO 2 I = 1,N
       H = 0.25*H
       D = SIN(X + H) - F
       Q = D/H
       E = ABS(G - Q)
       PRINT *,H,D,Q,E
2   CONTINUE
    STOP
END
```

A.1.3. Versión Iteración 2

```
program FIRST
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
    data  N/25/, H/1.0/, X/0.5/
    F = SIN(x)
    G = COS(x)
    do 2 I = 1,n
       H = 0.25*H
       D = SIN(X + H) - F
       Q = D/H
       E = ABS(G - Q)
       print *,H,D,Q,E
2   continue
    stop
end
```

A.1.4. Versión Iteración 3

```
program FIRST
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
    data  N/25/, H/1.0/, X/0.5/
    f = SIN(x)
    g = COS(x)
    do 2 i = 1,n
       h = 0.25*h
       d = SIN(x + h) - f
       q = d/h
       e = ABS(g - q)
       print *,h,d,q,e
2   continue
    stop
end
```

A.1.5. Versión Iteración 4

```
program FIRST
    implicit none
    real :: d
    real :: e
    real :: f
    real :: g
    real :: h
    integer :: i
    integer :: n
    real :: q
    real :: x
    !
    ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
    !
    ! FILE: first.f
    !
    ! FIRST PROGRAMMING EXPERIMENT
    !
    data  n/25/, h/1.0/, x/0.5/
    f = SIN(x)
    g = COS(x)
    do 2 i = 1,n
       h = 0.25*h
       d = SIN(x + h) - f
       q = d/h
       e = ABS(g - q)
       print *,h,d,q,e
2   continue
    stop
end
```

A.1.6. Versión Iteración 5

```
program FIRST
    implicit none
    real :: d
    real :: e
    real :: f
    real :: g
    real :: h
    integer :: i
    integer :: n
    real :: q
    real :: x
    !
    ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
    !
    ! FILE: first.f
    !
    ! FIRST PROGRAMMING EXPERIMENT
    !
    data  n/25/, h/1.0/, x/0.5/
    f = SIN(x)
    g = COS(x)
    do i = 1,n
       h = 0.25*h
       d = SIN(x + h) - f
       q = d/h
       e = ABS(g - q)
       print *,h,d,q,e
2   continue
    END DO
    stop
end
```

A.1.7. Versión Iteración 6

```
program FIRST
    implicit none
    real:: d,e,f,g,h,q,x
    integer :: i,n
    !
    ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
    !
    ! FILE: first.f
    !
    ! FIRST PROGRAMMING EXPERIMENT
    !
    data  n/25/, h/1.0/, x/0.5/
    f = SIN(x)
    g = COS(x)
    do i = 1,n
       h = 0.25*h
       d = SIN(x + h) - f
       q = d/h
       e = ABS(g - q)
       print *,h,d,q,e
      continue
    END DO
    stop
end
```

A.1.8. Versión Iteración 7

```
program FIRST
   implicit none
   real:: d,e,f,g,h,q,x
   integer :: i,n
   !
   ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
   !
   ! FILE: first.f
   !
   ! FIRST PROGRAMMING EXPERIMENT
   !
   data  n/25/, h/1.0/, x/0.5/
   f = SIN(x)
   g = COS(x)
   do i = 1,n
      h = 0.25*h
      d = SIN(x + h) - f
      q = d/h
      e = ABS(g - q)
      print *,h,d,q,e
   END DO
end
```

A.1.9. Versión Iteración 8

```
program FIRST
    implicit none
    real:: d,e,f,g,h,q,x
    integer :: i,n
    !
    ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
    !
    ! FILE: first.f
    !
    ! FIRST PROGRAMMING EXPERIMENT
    !
    data  n/25/, h/1.0/, x/0.5/
    f = SIN(x)
    g = COS(x)
    do i = 1,n
       h = 0.25*h
       d = SIN(x + h) - f
       q = d/h
       e = ABS(g - q)
       print *,h,d,q,e
    END DO
end program FIRST
```

A.1.10. Versión Iteración 9

```
program FIRST
    implicit none
    real:: d,e,f,g,h,q,x
    integer :: i,n
    !
    ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
    !
    ! FILE: first.f
    !
    ! FIRST PROGRAMMING EXPERIMENT
    !
    data  n /25/
    x=0.5
    h=1.0
    f = SIN(x)
    g = COS(x)
    do i = 1,n
       h = 0.25*h
       d = SIN(x + h) - f
       q = d/h
       e = ABS(g - q)
       print *,h,d,q,e
    END DO
end program FIRST
```

A.1.11. Versión Iteración 10

```
program FIRST
   implicit none
   real:: d,e,f,g,h,q,x
   integer :: i,n
   !
   ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
   !
   ! FILE: first.f
   !
   ! FIRST PROGRAMMING EXPERIMENT
   !
   parameter ( n = 25 )
   x=0.5
   h=1.0
   f = SIN(x)
   g = COS(x)
   do i = 1,n
      h = 0.25*h
      d = SIN(x + h) - f
      q = d/h
      e = ABS(g - q)
      print *,h,d,q,e
   END DO
end program FIRST
```

A.1.12. Versión Iteración 11

```
program FIRST
   implicit none
   real:: d,e,f,g,h,q,x
   integer :: i,n
   !
   ! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
   !
   ! FILE: first.f
   !
   ! FIRST PROGRAMMING EXPERIMENT
   !
   parameter ( n = 25 )
   x=0.5
   h=1.0
   f = SIN(x)
   g = COS(x)
   do i = 1,n
      h = 0.25*h
      d = SIN(x + h) - f
      q = d/h
      e = ABS(g - q)
      print *,h,d,q,e
   end do
end program FIRST
```

A.1.13. Versión Final

```
!
! NUMERICAL MATHEMATICS AND COMPUTING, CHENEY/KINCAID, (c) 1985
!
! FILE: first.f
!
! FIRST PROGRAMMING EXPERIMENT
!
program FIRST
   implicit none
   real:: d,e,f,g,h,q,x
   integer :: i,n
   parameter ( n = 25 )
   x=0.5
   h=1.0
   f = SIN(x)
   g = COS(x)
   do i = 1,n
      h = 0.25*h
      d = SIN(x + h) - f
      q = d/h
      e = ABS(g - q)
      print *,h,d,q,e
   end do
end program FIRST
```

A.1.14. Versión publicada en la edición 5 del libro re-escrita por el autor en Fortran 90

```
! Numerical Mathematics and Computing, Fifth Edition
! Ward Cheney & David Kincaid
! Brooks/Cole Publ. Co.
! Copyright (c) 2003.  All rights reserved.
! For educational use with the Cheney-Kincaid textbook.
! Absolutely no warranty implied or expressed.
!
! Section 1.1
!
! File: first.f90
!
! First programming experiment:  compute derivative of sin(x)
! by limit definition
program first
   integer, parameter :: n =25
   integer :: i
   real :: error, h, y

   x = 0.5
   h = 1.0
   print*, "  i   h   y   error"
   do i = 1,n
      h = 0.25*h
      y = (sin(x + h) - sin(x))/h
      error = abs(cos(x) - y)
      print *, i, h, y, error
  end do
end program first
```

A.1.15. Resultados Primera ejecución

```
0.250000000      0.202213228      0.808852911      6.87296391E-02
6.25000000E-02   5.38771152E-02   0.862033844      1.55487061E-02
1.56250000E-02   1.36531293E-02   0.873800278      3.78227234E-03
3.90625000E-03   3.42437625E-03   0.876640320      9.42230225E-04
9.76562500E-04   8.56786966E-04   0.877349854      2.32696533E-04
2.44140625E-04   2.14219093E-04   0.877441406      1.41143799E-04
6.10351562E-05   5.35547733E-05   0.877441406      1.41143799E-04
1.52587891E-05   1.33812428E-05   0.876953125      6.29425049E-04
3.81469727E-06   3.33786011E-06   0.875000000      2.58255005E-03
9.53674316E-07   8.34465027E-07   0.875000000      2.58255005E-03
2.38418579E-07   2.08616257E-07   0.875000000      2.58255005E-03
5.96046448E-08   2.98023224E-08   0.500000000     0.377582550
1.49011612E-08   0.00000000       0.00000000      0.877582550
3.72529030E-09   0.00000000       0.00000000      0.877582550
9.31322575E-10   0.00000000       0.00000000      0.877582550
2.32830644E-10   0.00000000       0.00000000      0.877582550
5.82076609E-11   0.00000000       0.00000000      0.877582550
1.45519152E-11   0.00000000       0.00000000      0.877582550
3.63797881E-12   0.00000000       0.00000000      0.877582550
9.09494702E-13   0.00000000       0.00000000      0.877582550
2.27373675E-13   0.00000000       0.00000000      0.877582550
5.68434189E-14   0.00000000       0.00000000      0.877582550
1.42108547E-14   0.00000000       0.00000000      0.877582550
3.55271368E-15   0.00000000       0.00000000      0.877582550
8.88178420E-16   0.00000000       0.00000000      0.877582550
```

A.1.16. Resultados Fin de la Iteración 1

```
0.250000000      0.202213228      0.808852911      6.87296391E-02
6.25000000E-02   5.38771152E-02   0.862033844      1.55487061E-02
1.56250000E-02   1.36531293E-02   0.873800278      3.78227234E-03
3.90625000E-03   3.42437625E-03   0.876640320      9.42230225E-04
9.76562500E-04   8.56786966E-04   0.877349854      2.32696533E-04
2.44140625E-04   2.14219093E-04   0.877441406      1.41143799E-04
6.10351562E-05   5.35547733E-05   0.877441406      1.41143799E-04
1.52587891E-05   1.33812428E-05   0.876953125      6.29425049E-04
3.81469727E-06   3.33786011E-06   0.875000000      2.58255005E-03
9.53674316E-07   8.34465027E-07   0.875000000      2.58255005E-03
2.38418579E-07   2.08616257E-07   0.875000000      2.58255005E-03
5.96046448E-08   2.98023224E-08   0.500000000     0.377582550
1.49011612E-08   0.00000000       0.00000000      0.877582550
3.72529030E-09   0.00000000       0.00000000      0.877582550
9.31322575E-10   0.00000000       0.00000000      0.877582550
2.32830644E-10   0.00000000       0.00000000      0.877582550
5.82076609E-11   0.00000000       0.00000000      0.877582550
1.45519152E-11   0.00000000       0.00000000      0.877582550
3.63797881E-12   0.00000000       0.00000000      0.877582550
9.09494702E-13   0.00000000       0.00000000      0.877582550
2.27373675E-13   0.00000000       0.00000000      0.877582550
5.68434189E-14   0.00000000       0.00000000      0.877582550
1.42108547E-14   0.00000000       0.00000000      0.877582550
3.55271368E-15   0.00000000       0.00000000      0.877582550
8.88178420E-16   0.00000000       0.00000000      0.877582550
```

A.1.17. Resultados Fin de la Iteración 2

0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

A.1.18. Resultados Fin de la Iteración 3

0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

A.1.19. Resultados Fin de la Iteración 4

```
0.250000000      0.202213228      0.808852911      6.87296391E-02
6.25000000E-02   5.38771152E-02   0.862033844      1.55487061E-02
1.56250000E-02   1.36531293E-02   0.873800278      3.78227234E-03
3.90625000E-03   3.42437625E-03   0.876640320      9.42230225E-04
9.76562500E-04   8.56786966E-04   0.877349854      2.32696533E-04
2.44140625E-04   2.14219093E-04   0.877441406      1.41143799E-04
6.10351562E-05   5.35547733E-05   0.877441406      1.41143799E-04
1.52587891E-05   1.33812428E-05   0.876953125      6.29425049E-04
3.81469727E-06   3.33786011E-06   0.875000000      2.58255005E-03
9.53674316E-07   8.34465027E-07   0.875000000      2.58255005E-03
2.38418579E-07   2.08616257E-07   0.875000000      2.58255005E-03
5.96046448E-08   2.98023224E-08   0.500000000     0.377582550
1.49011612E-08   0.00000000       0.00000000      0.877582550
3.72529030E-09   0.00000000       0.00000000      0.877582550
9.31322575E-10   0.00000000       0.00000000      0.877582550
2.32830644E-10   0.00000000       0.00000000      0.877582550
5.82076609E-11   0.00000000       0.00000000      0.877582550
1.45519152E-11   0.00000000       0.00000000      0.877582550
3.63797881E-12   0.00000000       0.00000000      0.877582550
9.09494702E-13   0.00000000       0.00000000      0.877582550
2.27373675E-13   0.00000000       0.00000000      0.877582550
5.68434189E-14   0.00000000       0.00000000      0.877582550
1.42108547E-14   0.00000000       0.00000000      0.877582550
3.55271368E-15   0.00000000       0.00000000      0.877582550
8.88178420E-16   0.00000000       0.00000000      0.877582550
```

A.1.20. Resultados Fin de la Iteración 5

```
0.250000000      0.202213228      0.808852911      6.87296391E-02
6.25000000E-02   5.38771152E-02   0.862033844      1.55487061E-02
1.56250000E-02   1.36531293E-02   0.873800278      3.78227234E-03
3.90625000E-03   3.42437625E-03   0.876640320      9.42230225E-04
9.76562500E-04   8.56786966E-04   0.877349854      2.32696533E-04
2.44140625E-04   2.14219093E-04   0.877441406      1.41143799E-04
6.10351562E-05   5.35547733E-05   0.877441406      1.41143799E-04
1.52587891E-05   1.33812428E-05   0.876953125      6.29425049E-04
3.81469727E-06   3.33786011E-06   0.875000000      2.58255005E-03
9.53674316E-07   8.34465027E-07   0.875000000      2.58255005E-03
2.38418579E-07   2.08616257E-07   0.875000000      2.58255005E-03
5.96046448E-08   2.98023224E-08   0.500000000     0.377582550
1.49011612E-08   0.00000000       0.00000000      0.877582550
3.72529030E-09   0.00000000       0.00000000      0.877582550
9.31322575E-10   0.00000000       0.00000000      0.877582550
2.32830644E-10   0.00000000       0.00000000      0.877582550
5.82076609E-11   0.00000000       0.00000000      0.877582550
1.45519152E-11   0.00000000       0.00000000      0.877582550
3.63797881E-12   0.00000000       0.00000000      0.877582550
9.09494702E-13   0.00000000       0.00000000      0.877582550
2.27373675E-13   0.00000000       0.00000000      0.877582550
5.68434189E-14   0.00000000       0.00000000      0.877582550
1.42108547E-14   0.00000000       0.00000000      0.877582550
3.55271368E-15   0.00000000       0.00000000      0.877582550
8.88178420E-16   0.00000000       0.00000000      0.877582550
```

A.1.21. Resultados Fin de la Iteración 6

0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

A.1.22. Resultados Fin de la Iteración 7

0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

A.1.23. Resultados Fin de la Iteración 8

```
0.250000000      0.202213228      0.808852911      6.87296391E-02
6.25000000E-02   5.38771152E-02   0.862033844      1.55487061E-02
1.56250000E-02   1.36531293E-02   0.873800278      3.78227234E-03
3.90625000E-03   3.42437625E-03   0.876640320      9.42230225E-04
9.76562500E-04   8.56786966E-04   0.877349854      2.32696533E-04
2.44140625E-04   2.14219093E-04   0.877441406      1.41143799E-04
6.10351562E-05   5.35547733E-05   0.877441406      1.41143799E-04
1.52587891E-05   1.33812428E-05   0.876953125      6.29425049E-04
3.81469727E-06   3.33786011E-06   0.875000000      2.58255005E-03
9.53674316E-07   8.34465027E-07   0.875000000      2.58255005E-03
2.38418579E-07   2.08616257E-07   0.875000000      2.58255005E-03
5.96046448E-08   2.98023224E-08   0.500000000     0.377582550
1.49011612E-08   0.00000000       0.00000000      0.877582550
3.72529030E-09   0.00000000       0.00000000      0.877582550
9.31322575E-10   0.00000000       0.00000000      0.877582550
2.32830644E-10   0.00000000       0.00000000      0.877582550
5.82076609E-11   0.00000000       0.00000000      0.877582550
1.45519152E-11   0.00000000       0.00000000      0.877582550
3.63797881E-12   0.00000000       0.00000000      0.877582550
9.09494702E-13   0.00000000       0.00000000      0.877582550
2.27373675E-13   0.00000000       0.00000000      0.877582550
5.68434189E-14   0.00000000       0.00000000      0.877582550
1.42108547E-14   0.00000000       0.00000000      0.877582550
3.55271368E-15   0.00000000       0.00000000      0.877582550
8.88178420E-16   0.00000000       0.00000000      0.877582550
```

A.1.24. Resultados Fin de la Iteración 9

```
0.250000000      0.202213228      0.808852911      6.87296391E-02
6.25000000E-02   5.38771152E-02   0.862033844      1.55487061E-02
1.56250000E-02   1.36531293E-02   0.873800278      3.78227234E-03
3.90625000E-03   3.42437625E-03   0.876640320      9.42230225E-04
9.76562500E-04   8.56786966E-04   0.877349854      2.32696533E-04
2.44140625E-04   2.14219093E-04   0.877441406      1.41143799E-04
6.10351562E-05   5.35547733E-05   0.877441406      1.41143799E-04
1.52587891E-05   1.33812428E-05   0.876953125      6.29425049E-04
3.81469727E-06   3.33786011E-06   0.875000000      2.58255005E-03
9.53674316E-07   8.34465027E-07   0.875000000      2.58255005E-03
2.38418579E-07   2.08616257E-07   0.875000000      2.58255005E-03
5.96046448E-08   2.98023224E-08   0.500000000     0.377582550
1.49011612E-08   0.00000000       0.00000000      0.877582550
3.72529030E-09   0.00000000       0.00000000      0.877582550
9.31322575E-10   0.00000000       0.00000000      0.877582550
2.32830644E-10   0.00000000       0.00000000      0.877582550
5.82076609E-11   0.00000000       0.00000000      0.877582550
1.45519152E-11   0.00000000       0.00000000      0.877582550
3.63797881E-12   0.00000000       0.00000000      0.877582550
9.09494702E-13   0.00000000       0.00000000      0.877582550
2.27373675E-13   0.00000000       0.00000000      0.877582550
5.68434189E-14   0.00000000       0.00000000      0.877582550
1.42108547E-14   0.00000000       0.00000000      0.877582550
3.55271368E-15   0.00000000       0.00000000      0.877582550
8.88178420E-16   0.00000000       0.00000000      0.877582550
```

A.1.25. Resultados Fin de la Iteración 10

0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

A.1.26. Resultados Fin de la Iteración 11

0.250000000	0.202213228	0.808852911	6.87296391E-02
6.25000000E-02	5.38771152E-02	0.862033844	1.55487061E-02
1.56250000E-02	1.36531293E-02	0.873800278	3.78227234E-03
3.90625000E-03	3.42437625E-03	0.876640320	9.42230225E-04
9.76562500E-04	8.56786966E-04	0.877349854	2.32696533E-04
2.44140625E-04	2.14219093E-04	0.877441406	1.41143799E-04
6.10351562E-05	5.35547733E-05	0.877441406	1.41143799E-04
1.52587891E-05	1.33812428E-05	0.876953125	6.29425049E-04
3.81469727E-06	3.33786011E-06	0.875000000	2.58255005E-03
9.53674316E-07	8.34465027E-07	0.875000000	2.58255005E-03
2.38418579E-07	2.08616257E-07	0.875000000	2.58255005E-03
5.96046448E-08	2.98023224E-08	0.500000000	0.377582550
1.49011612E-08	0.00000000	0.00000000	0.877582550
3.72529030E-09	0.00000000	0.00000000	0.877582550
9.31322575E-10	0.00000000	0.00000000	0.877582550
2.32830644E-10	0.00000000	0.00000000	0.877582550
5.82076609E-11	0.00000000	0.00000000	0.877582550
1.45519152E-11	0.00000000	0.00000000	0.877582550
3.63797881E-12	0.00000000	0.00000000	0.877582550
9.09494702E-13	0.00000000	0.00000000	0.877582550
2.27373675E-13	0.00000000	0.00000000	0.877582550
5.68434189E-14	0.00000000	0.00000000	0.877582550
1.42108547E-14	0.00000000	0.00000000	0.877582550
3.55271368E-15	0.00000000	0.00000000	0.877582550
8.88178420E-16	0.00000000	0.00000000	0.877582550

Apéndice B

Código Fuente y Resultados de: Caso de Estudio 2, Caso de Estudio 3 y Caso de Estudio 4

B.1. Programa: geokerr.f

Código Fuente del programa geokerr.f a través de las iteraciones:

B.1.1. Versiones

Debido a la extensa longitud del programa se ha decidido colocar los enlace al versionador de código fuente https://bitbucket.org/marianomendez/geokerr/src.

B.1.2. Formato de los Resultados

Extraído del archivo geokerr_readme se adjunta definición textual de los datos de salida:

To reproduce Fig. 1 of Dexter & Agol (2009), plot dt vs. uf for each geodesic.

OUTPUT FORMAT

The first line in the output file contains a summary of the rest of the file, including the total number of geodesics, mu0, a, and u0. Then each geodesic is written with the first line specifying alpha, beta (or q2, l if provided instead), the number of points on the geodesic, and the case number corresponding to Tables 1,2 of Dexter & Agol (2009). Finally, the geodesic coordinates are output with the columns uf, muf, dt, dphi, lambda, mu turning points, u turning points. A template is given here:

```
NGEO MUO A UO
ALPHA BETA NUP NCASE
UFI(1) MUFI(1) DTI(1)        DPHI(1) LAMBDAI(1)            TPMI(1) TPRI(1)
UFI(2) MUFI(2) DTI(2)-DTI(1) DPHI(2) LAMBDAI(2)-LAMBDAI(1) TPMI(2) TPRI(2)
UFI(3) MUFI(3) DTI(3)-DTI(1) DPHI(3) LAMBDAI(3)-LAMBDAI(1) TPMI(3) TPRI(3)
...
ALPHA2 BETA2 NUP2 NCASE2
...
```

B.1.3. Resultados de la Versión Original

```
100   0.0000000000000000       0.99800000000000000       1.8545994065281900E-006
-3.3999999999999999       -5.4000000000000004                1   1.8545994065281900E-006           7
1.85841684E-06   6.82638462E-01   1.07734712E+06  3.95031134E+00   1.07729503E+06 1 1
-3.3999999999999999       -4.2000000000000002                1   1.8545994065281900E-006           7
1.85928834E-06   7.15395217E-01   1.07709656E+06  4.38960416E+00   1.07704286E+06 1 1
-3.3999999999999999       -3.0000000000000000                1   1.8545994065281900E-006           7
1.86044847E-06   6.62867487E-01   1.07676312E+06  4.98688108E+00   1.07670749E+06 1 1
-3.3999999999999999       -1.7999999999999998                1   1.8545994065281900E-006           7
1.86178526E-06   4.78814069E-01   1.07637909E+06  5.55249559E+00   1.07632147E+06 2 1
-3.3999999999999999       -0.59999999999999964               1   1.8545994065281900E-006           7
1.86278867E-06   1.78016854E-01   1.07609095E+06  5.88462484E+00   1.07603199E+06 2 1
-3.3999999999999999        0.59999999999999964               1   1.8545994065281900E-006           7
1.86278867E-06  -1.78016854E-01   1.07609095E+06  5.88462484E+00   1.07603199E+06 2 1
-3.3999999999999999        1.7999999999999998                1   1.8545994065281900E-006           7
1.86178526E-06  -4.78814069E-01   1.07637909E+06  5.55249559E+00   1.07632147E+06 2 1
-3.3999999999999999        3.0000000000000000                1   1.8545994065281900E-006           7
1.86044847E-06  -6.62867487E-01   1.07676312E+06  4.98688108E+00   1.07670749E+06 1 1
-3.3999999999999999        4.1999999999999993                1   1.8545994065281900E-006           7
1.85928834E-06  -7.15395217E-01   1.07709656E+06  4.38960416E+00   1.07704286E+06 1 1
-3.3999999999999999        5.4000000000000004                1   1.8545994065281900E-006           7
1.85841684E-06  -6.82638462E-01   1.07734712E+06  3.95031134E+00   1.07729503E+06 1 1
-2.2000000000000002       -5.4000000000000004                1   1.8545994065281900E-006           7
1.85895957E-06   8.64438018E-01   1.07719187E+06  4.17689778E+00   1.07713830E+06 1 1
-2.2000000000000002       -4.2000000000000002                1   1.8545994065281900E-006           7
1.86039870E-06   8.79666770E-01   1.07677911E+06  5.57903208E+00   1.07672253E+06 2 1
-2.2000000000000002       -3.0000000000000000                1   1.8545994065281900E-006           7
1.86302498E-06   2.63473992E-01   1.07602793E+06  7.52794496E+00   1.07596536E+06 2 1
-2.2000000000000002       -1.7999999999999998                1   1.8545994065281900E-006           7
1.86769409E-06  -6.38920066E-01   1.07470250E+06  1.38935726E+01   1.07462416E+06 3 1
-2.2000000000000002       -0.59999999999999964               1   1.8545994065281900E-006           7
1.87106767E-06  -2.32529984E-01   1.07375702E+06  2.27587043E+01   1.07365820E+06 3 1
-2.2000000000000002        0.59999999999999964               1   1.8545994065281900E-006           7
1.87106767E-06   2.32529984E-01   1.07375702E+06  2.27587043E+01   1.07365820E+06 3 1
-2.2000000000000002        1.7999999999999998                1   1.8545994065281900E-006           7
1.86769409E-06   6.38920066E-01   1.07470250E+06  1.38935726E+01   1.07462416E+06 3 1
-2.2000000000000002        3.0000000000000000                1   1.8545994065281900E-006           7
1.86302498E-06  -2.63473992E-01   1.07602793E+06  7.52794496E+00   1.07596536E+06 2 1
-2.2000000000000002        4.1999999999999993                1   1.8545994065281900E-006           7
1.86039870E-06  -8.79666770E-01   1.07677911E+06  5.57903208E+00   1.07672253E+06 2 1
-2.2000000000000002        5.4000000000000004                1   1.8545994065281900E-006           7
1.85895957E-06  -8.64438018E-01   1.07719187E+06  4.17689778E+00   1.07713830E+06 1 1
-1.0000000000000000       -5.4000000000000004                1   1.8545994065281900E-006           7
1.85950828E-06   9.82996990E-01   1.07703569E+06  5.33775328E+00   1.07698025E+06 2 1
-1.0000000000000000       -4.2000000000000002                1   1.8545994065281900E-006           7
1.86242860E-06  -7.10903642E-01   1.07620483E+06  8.19824256E+00   1.07613933E+06 2 1
-1.0000000000000000       -3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01   5.46989029E-01   5.39511482E+05  1.32529223E+02   5.39201348E+05 1 0
-1.0000000000000000       -1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01  -7.60368883E-01   5.39506117E+05  1.30713322E+02   5.39199995E+05 1 0
-1.0000000000000000       -0.59999999999999964               1   1.8545994065281900E-006           5
```

```
9.40544484E-01  -5.23111545E-01   5.39503793E+05   1.28766321E+02   5.39199092E+05 0 0
-1.0000000000000000        0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01   5.23111545E-01   5.39503793E+05   1.28766321E+02   5.39199092E+05 0 0
-1.0000000000000000         1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01   7.60368883E-01   5.39506117E+05   1.30713322E+02   5.39199995E+05 1 0
-1.0000000000000000         3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01  -5.46989029E-01   5.39511482E+05   1.32529223E+02   5.39201348E+05 1 0
-1.0000000000000000         4.1999999999999993                1   1.8545994065281900E-006           7
1.86242860E-06   7.10903642E-01   1.07620483E+06   8.19824256E+00   1.07613933E+06 2 1
-1.0000000000000000         5.4000000000000004                1   1.8545994065281900E-006           7
1.85950828E-06  -9.82996990E-01   1.07703569E+06   5.33775328E+00   1.07698025E+06 2 1
0.20000000000000018        -5.4000000000000004                1   1.8545994065281900E-006           7
1.85997039E-06   7.79611930E-01   1.07690564E+06  -5.64547518E+00   1.07684781E+06 2 1
0.20000000000000018        -4.2000000000000002                1   1.8545994065281900E-006           5
9.40544484E-01   9.71245339E-01   5.39514301E+05   1.26021712E+02   5.39202419E+05 1 0
0.20000000000000018        -3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01  -5.01381867E-01   5.39507532E+05   1.25375286E+02   5.39200160E+05 1 0
0.20000000000000018        -1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01  -9.92075552E-01   5.39504702E+05   1.27223824E+02   5.39199061E+05 0 0
0.20000000000000018        -0.59999999999999964               1   1.8545994065281900E-006           5
9.40544484E-01  -4.67463319E-01   5.39503293E+05   1.27800415E+02   5.39198354E+05 0 0
0.20000000000000018         0.59999999999999964               1   1.8545994065281900E-006           5
9.40544484E-01   4.67463319E-01   5.39503293E+05   1.27800415E+02   5.39198354E+05 0 0
0.20000000000000018         1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01   9.92075552E-01   5.39504702E+05   1.27223824E+02   5.39199061E+05 0 0
0.20000000000000018         3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01   5.01381867E-01   5.39507532E+05   1.25375286E+02   5.39200160E+05 1 0
0.20000000000000018         4.1999999999999993                1   1.8545994065281900E-006           5
9.40544484E-01  -9.71245339E-01   5.39514301E+05   1.26021712E+02   5.39202419E+05 1 0
0.20000000000000018         5.4000000000000004                1   1.8545994065281900E-006           7
1.85997039E-06  -7.79611930E-01   1.07690564E+06  -5.64547518E+00   1.07684781E+06 2 1
1.4000000000000004         -5.4000000000000004                1   1.8545994065281900E-006           7
1.86014972E-06   1.37158112E-01   1.07685780E+06  -5.46118593E+00   1.07679751E+06 2 1
1.4000000000000004         -4.2000000000000002                1   1.8545994065281900E-006           5
9.40544484E-01   6.00138751E-01   5.39511730E+05   1.25557621E+02   5.39201617E+05 1 0
1.4000000000000004         -3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01  -5.76087522E-01   5.39507048E+05   1.25627110E+02   5.39199795E+05 1 0
1.4000000000000004         -1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01  -8.15771227E-01   5.39504782E+05   1.26640492E+02   5.39198772E+05 0 0
1.4000000000000004        -0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01  -3.57392479E-01   5.39503689E+05   1.27190798E+02   5.39198198E+05 0 0
1.4000000000000004         0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01   3.57392479E-01   5.39503689E+05   1.27190798E+02   5.39198198E+05 0 0
1.4000000000000004         1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01   8.15771227E-01   5.39504782E+05   1.26640492E+02   5.39198772E+05 0 0
1.4000000000000004         3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01   5.76087522E-01   5.39507048E+05   1.25627110E+02   5.39199795E+05 1 0
1.4000000000000004         4.1999999999999993                1   1.8545994065281900E-006           5
9.40544484E-01  -6.00138751E-01   5.39511730E+05   1.25557621E+02   5.39201617E+05 1 0
1.4000000000000004         5.4000000000000004                1   1.8545994065281900E-006           7
1.86014972E-06  -1.37158112E-01   1.07685780E+06  -5.46118593E+00   1.07679751E+06 2 1
2.5999999999999996        -5.4000000000000004                1   1.8545994065281900E-006           7
1.85971516E-06   1.87939424E-01   1.07698322E+06  -5.50392066E+00   1.07692353E+06 2 1
2.5999999999999996        -4.2000000000000002                1   1.8545994065281900E-006           5
9.40544484E-01   6.15908703E-01   5.39512331E+05   1.25298421E+02   5.39201922E+05 1 0
2.5999999999999996        -3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01  -3.18670707E-01   5.39507789E+05   1.25685903E+02   5.39199976E+05 1 0
2.5999999999999996        -1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01  -5.40800621E-01   5.39505663E+05   1.26275661E+02   5.39198970E+05 1 0
2.5999999999999996       -0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01  -2.41257185E-01   5.39504688E+05   1.26675232E+02   5.39198462E+05 0 0
2.5999999999999996        0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01   2.41257185E-01   5.39504688E+05   1.26675232E+02   5.39198462E+05 0 0
2.5999999999999996        1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01   5.40800621E-01   5.39505663E+05   1.26275661E+02   5.39198970E+05 1 0
2.5999999999999996        3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01   3.18670707E-01   5.39507789E+05   1.25685903E+02   5.39199976E+05 1 0
2.5999999999999996        4.1999999999999993                1   1.8545994065281900E-006           5
9.40544484E-01  -6.15908703E-01   5.39512331E+05   1.25298421E+02   5.39201922E+05 1 0
2.5999999999999996        5.4000000000000004                1   1.8545994065281900E-006           7
1.85971516E-06  -1.87939424E-01   1.07698322E+06  -5.50392066E+00   1.07692353E+06 2 1
3.7999999999999998        -5.4000000000000004                1   1.8545994065281900E-006           7
1.85892248E-06   6.86068901E-01   1.07720844E+06  -5.14437278E+00   1.07715180E+06 2 1
3.7999999999999998        -4.2000000000000002                1   1.8545994065281900E-006           5
9.40544484E-01   7.45320893E-01   5.39515581E+05   1.24482405E+02   5.39203408E+05 1 0
3.7999999999999998        -3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01   7.46166264E-02   5.39509716E+05   1.25461937E+02   5.39200789E+05 1 0
3.7999999999999998        -1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01  -2.28620198E-01   5.39507372E+05   1.25875945E+02   5.39199671E+05 1 0
3.7999999999999998       -0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01  -1.23714970E-01   5.39506371E+05   1.26157951E+02   5.39199162E+05 1 0
3.7999999999999998        0.59999999999999964                1   1.8545994065281900E-006           5
9.40544484E-01   1.23714970E-01   5.39506371E+05   1.26157951E+02   5.39199162E+05 1 0
3.7999999999999998        1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01   2.28620198E-01   5.39507372E+05   1.25875945E+02   5.39199671E+05 1 0
3.7999999999999998        3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01  -7.46166264E-02   5.39509716E+05   1.25461937E+02   5.39200789E+05 1 0
3.7999999999999998        4.1999999999999993                1   1.8545994065281900E-006           5
9.40544484E-01  -7.45320893E-01   5.39515581E+05   1.24482405E+02   5.39203408E+05 1 0
3.7999999999999998        5.4000000000000004                1   1.8545994065281900E-006           7
1.85892248E-06  -6.86068901E-01   1.07720844E+06  -5.14437278E+00   1.07715180E+06 2 1
5.0000000000000000        -5.4000000000000004                1   1.8545994065281900E-006           7
1.85821825E-06   7.36766887E-01   1.07740885E+06  -4.43110163E+00   1.07735474E+06 1 1
5.0000000000000000        -4.2000000000000002                1   1.8545994065281900E-006           7
1.85964679E-06  -6.27648457E-01   1.07701149E+06  -6.55120151E+00   1.07694728E+06 2 1
5.0000000000000000        -3.0000000000000000                1   1.8545994065281900E-006           5
9.40544484E-01   4.60548128E-01   5.39513849E+05   1.24779779E+02   5.39202721E+05 1 0
5.0000000000000000        -1.7999999999999998                1   1.8545994065281900E-006           5
9.40544484E-01   9.63450863E-02   5.39510415E+05   1.25319609E+02   5.39201103E+05 1 0
5.0000000000000000        -0.59999999999999964               1   1.8545994065281900E-006           5
9.40544484E-01  -5.17788598E-03   5.39509142E+05   1.25545845E+02   5.39200478E+05 1 0
5.0000000000000000         0.59999999999999964               1   1.8545994065281900E-006           5
9.40544484E-01   5.17788598E-03   5.39509142E+05   1.25545845E+02   5.39200478E+05 1 0
```

```
5.0000000000000000        1.7999999999999998                1   1.8545994065281900E-006          5
9.40544484E-01  -9.63450863E-02   5.39510415E+05   1.25319609E+02   5.39201103E+05 1 0
5.0000000000000000        3.0000000000000000                1   1.8545994065281900E-006          5
9.40544484E-01  -4.60548128E-01   5.39513849E+05   1.24779779E+02   5.39202721E+05 1 0
5.0000000000000000        4.1999999999999993                1   1.8545994065281900E-006          7
1.85964679E-06   6.27648457E-01   1.07701149E+06  -6.55120151E+00   1.07694728E+06 2 1
5.0000000000000000        5.4000000000000004                1   1.8545994065281900E-006          7
1.85821825E-06  -7.36766887E-01   1.07740885E+06  -4.43110163E+00   1.07735474E+06 1 1
6.1999999999999993        -5.4000000000000004               1   1.8545994065281900E-006          7
1.85766990E-06   6.08222309E-01   1.07756525E+06  -4.03724702E+00   1.07751296E+06 1 1
6.1999999999999993        -4.2000000000000002               1   1.8545994065281900E-006          7
1.85821994E-06   5.59591422E-01   1.07740934E+06  -4.59174694E+00   1.07735475E+06 2 1
6.1999999999999993        -3.0000000000000000               1   1.8545994065281900E-006          7
1.85918203E-06  -1.83366237E-01   1.07714147E+06  -5.97000200E+00   1.07708033E+06 2 1
6.1999999999999993        -1.7999999999999998               1   1.8545994065281900E-006          5
9.40544484E-01   2.70638447E-01   5.39517678E+05   1.24095948E+02   5.39204568E+05 2 0
6.1999999999999993        -0.59999999999999964              1   1.8545994065281900E-006          5
9.40544484E-01   9.24985904E-02   5.39514754E+05   1.24565355E+02   5.39203209E+05 1 0
6.1999999999999993        0.59999999999999964               1   1.8545994065281900E-006          5
9.40544484E-01  -9.24985904E-02   5.39514754E+05   1.24565355E+02   5.39203209E+05 1 0
6.1999999999999993        1.7999999999999998                1   1.8545994065281900E-006          5
9.40544484E-01  -2.70638447E-01   5.39517678E+05   1.24095948E+02   5.39204568E+05 2 0
6.1999999999999993        3.0000000000000000                1   1.8545994065281900E-006          7
1.85918203E-06   1.83366237E-01   1.07714147E+06  -5.97000200E+00   1.07708033E+06 2 1
6.1999999999999993        4.1999999999999993                1   1.8545994065281900E-006          7
1.85821994E-06  -5.59591422E-01   1.07740934E+06  -4.59174694E+00   1.07735475E+06 2 1
6.1999999999999993        5.4000000000000004                1   1.8545994065281900E-006          7
1.85766990E-06  -6.08222309E-01   1.07756525E+06  -4.03724702E+00   1.07751296E+06 1 1
7.4000000000000004        -5.4000000000000004               1   1.8545994065281900E-006          7
1.85724766E-06   4.72605032E-01   1.07768583E+06  -3.84669816E+00   1.07763492E+06 1 1
7.4000000000000004        -4.2000000000000002               1   1.8545994065281900E-006          7
1.85755767E-06   4.53470648E-01   1.07759767E+06  -4.08337465E+00   1.07754557E+06 1 1
7.4000000000000004        -3.0000000links000000000                1   1.8545994065281900E-006          7
1.85789706E-06   3.76274917E-01   1.07750147E+06  -4.38446577E+00   1.07744794E+06 1 1
7.4000000000000004        -1.7999999999999998               1   1.8545994065281900E-006          7
1.85823182E-06   2.27136179E-01   1.07740708E+06  -4.72718440E+00   1.07735190E+06 2 1
7.4000000000000004        -0.59999999999999964              1   1.8545994065281900E-006          7
1.85846988E-06   6.31102189E-02   1.07734040E+06  -5.01964037E+00   1.07728383E+06 2 1
7.4000000000000004        0.59999999999999964               1   1.8545994065281900E-006          7
1.85846988E-06  -6.31102189E-02   1.07734040E+06  -5.01964037E+00   1.07728383E+06 2 1
7.4000000000000004        1.7999999999999998                1   1.8545994065281900E-006          7
1.85823182E-06  -2.27136179E-01   1.07740708E+06  -4.72718440E+00   1.07735190E+06 2 1
7.4000000000000004        3.0000000000000000                1   1.8545994065281900E-006          7
1.85789706E-06  -3.76274917E-01   1.07750147E+06  -4.38446577E+00   1.07744794E+06 1 1
7.4000000000000004        4.1999999999999993                1   1.8545994065281900E-006          7
1.85755767E-06  -4.53470648E-01   1.07759767E+06  -4.08337465E+00   1.07754557E+06 1 1
7.4000000000000004        5.4000000000000004                1   1.8545994065281900E-006          7
1.85724766E-06  -4.72605032E-01   1.07768583E+06  -3.84669816E+00   1.07763492E+06 1 1
```

B.1.4. Resultados de las Subsiguiente Versiones

https://bitbucket.org/marianomendez/geokerr/src/

B.2. Ejemplo 2: geokerr.f90

A continuación se listan porciones de código fuente utilizadas en la iteración 5 de dicho ejemplo.

B.2.1. Perfilado del Programa geokerr.f90

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
40.00      0.06     0.06   482506     0.12     0.12  rc_
20.00      0.09     0.03    44784     0.67     0.67  rd_
13.33      0.11     0.02   118272     0.17     0.17  laguer_
6.67       0.12     0.01    88768     0.11     0.11  rf_
6.67       0.13     0.01    31296     0.32     0.96  ellquarticcomplex_
6.67       0.14     0.01    20000     0.50     3.22  geomu_
6.67       0.15     0.01    20000     0.50     0.50  sncndn_
```

```
0.00      0.15     0.00    74352     0.00     0.73  rj_
0.00      0.15     0.00    52624     0.00     0.62  ellquarticreal_
0.00      0.15     0.00    14784     0.00     0.23  calcimusym_
0.00      0.15     0.00    14784     0.00     1.35  zroots_
0.00      0.15     0.00    10000     0.00     4.20  calcphitmusym_
0.00      0.15     0.00    10000     0.00    15.00  geokerr_
0.00      0.15     0.00    10000     0.00     8.56  geophitime_

Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 6.67% of 0.15 seconds

index % time    self  children    called     name
0.00    0.15   10000/10000       MAIN__ [2]
[1]    100.0    0.00    0.15   10000         geokerr_ [1]
0.00    0.09   10000/10000       geophitime_ [3]
0.01    0.05   20000/20000       geomu_ [4]
-----------------------------------------------
<spontaneous>
[2]    100.0    0.00    0.15                 MAIN__ [2]
0.00    0.15   10000/10000       geokerr_ [1]
-----------------------------------------------
0.00    0.09   10000/10000       geokerr_ [1]
[3]     57.0    0.00    0.09   10000         geophitime_ [3]
0.00    0.04   10000/10000       calcphitmusym_ [7]
0.00    0.02   38272/52624       ellquarticreal_ [8]
0.01    0.01   20864/31296       ellquarticcomplex_ [10]
-----------------------------------------------
0.01    0.05   20000/20000       geokerr_ [1]
[4]     43.0    0.01    0.05   20000         geomu_ [4]
0.00    0.02   14784/14784       zroots_ [12]
0.01    0.00   20000/20000       sncndn_ [14]
0.00    0.01   10432/31296       ellquarticcomplex_ [10]
0.00    0.01   14352/52624       ellquarticreal_ [8]
0.00    0.00   14784/14784       calcimusym_ [15]
0.00    0.00   20000/88768       rf_ [13]
-----------------------------------------------
0.00    0.00   28704/482506      ellquarticreal_ [8]
0.00    0.00   31296/482506      ellquarticcomplex_ [10]
0.05    0.00  422506/482506      rj_ [6]
[5]     40.0    0.06    0.00  482506         rc_ [5]
-----------------------------------------------
0.00    0.01   15648/74352       ellquarticcomplex_ [10]
0.00    0.02   28704/74352       ellquarticreal_ [8]
0.00    0.02   30000/74352       calcphitmusym_ [7]
[6]     36.1    0.00    0.05   74352         rj_ [6]
0.05    0.00  422506/482506      rc_ [5]
0.00    0.00   14416/88768       rf_ [13]
-----------------------------------------------
0.00    0.04   10000/10000       geophitime_ [3]
[7]     28.0    0.00    0.04   10000         calcphitmusym_ [7]
0.00    0.02   30000/74352       rj_ [6]
0.02    0.00   30000/44784       rd_ [9]
-----------------------------------------------
0.00    0.01   14352/52624       geomu_ [4]
0.00    0.02   38272/52624       geophitime_ [3]
[8]     21.7    0.00    0.03   52624         ellquarticreal_ [8]
0.00    0.02   28704/74352       rj_ [6]
0.01    0.00    9568/44784       rd_ [9]
0.00    0.00   28704/482506      rc_ [5]
```

```
0.00    0.00   14352/88768       rf_ [13]
-----------------------------------------------
0.00    0.00    5216/44784       ellquarticcomplex_ [10]
0.01    0.00    9568/44784       ellquarticreal_ [8]
0.02    0.00   30000/44784       calcphitmusym_ [7]
[9]     20.0    0.03    0.00   44784         rd_ [9]
-----------------------------------------------
0.00    0.01   10432/31296       geomu_ [4]
0.01    0.01   20864/31296       geophitime_ [3]
[10]    20.0    0.01    0.02   31296         ellquarticcomplex_ [10]
0.00    0.01   15648/74352       rj_ [6]
0.00    0.00   31296/482506      rc_ [5]
0.00    0.00    5216/44784       rd_ [9]
0.00    0.00   10432/88768       rf_ [13]
-----------------------------------------------
0.02    0.00  118272/118272      zroots_ [12]
[11]    13.3    0.02    0.00  118272         laguer_ [11]
-----------------------------------------------
0.00    0.02   14784/14784       geomu_ [4]
[12]    13.3    0.00    0.02   14784         zroots_ [12]
0.02    0.00  118272/118272      laguer_ [11]
-----------------------------------------------
0.00    0.00   10432/88768       ellquarticcomplex_ [10]
0.00    0.00   14352/88768       ellquarticreal_ [8]
0.00    0.00   14416/88768       rj_ [6]
0.00    0.00   20000/88768       geomu_ [4]
0.00    0.00   29568/88768       calcimusym_ [15]
[13]     6.7    0.01    0.00   88768         rf_ [13]
-----------------------------------------------
0.01    0.00   20000/20000       geomu_ [4]
[14]     6.7    0.01    0.00   20000         sncndn_ [14]
-----------------------------------------------
0.00    0.00   14784/14784       geomu_ [4]
[15]     2.2    0.00    0.00   14784         calcimusym_ [15]
0.00    0.00   29568/88768       rf_ [13]
-----------------------------------------------

Index by function name

[15] calcimusym_             [4] geomu_                 [13] rf_
[7] calcphitmusym_           [3] geophitime_            [6] rj_
[10] ellquarticcomplex_      [11] laguer_               [14] sncndn_
[8] ellquarticreal_          [5] rc_                    [12] zroots_
[1] geokerr_                 [9] rd_
```

B.2.2. Lista de Funciones Candidatas a la Paralelización de geokerr.f90

Subrutina LAGUER()

```
subroutine LAGUER(a,m,x,its)
!************************************************************************
!     PURPOSE:  Find one root of a polynomial.
!     ARGUMENTS:
!     ROUTINES CALLED:
!     ALGORITHM:
!     ACCURACY:
!     REMARKS:  I don't have the documentation for this routine!
!     AUTHOR:  Press et al (1992)
!     DATE WRITTEN:  25 Mar 91.
!************************************************************************
implicit none
integer its,m
complex*16 a(m+1),x
!
integer iter,j,maxit,mr,mt
parameter (mr=8,mt=10,maxit=MT*MR)
double precision abx,abp,abm,err,epss,frac(mr)
parameter (epss=1.d-15)
complex*16 dx,x1,b,d,f,g,h,sq,gp,gm,g2
data FRAC /0.5d0,0.25d0,0.75d0,0.13d0,0.38d0,0.62d0,0.88d0,1.d0/
!       Loop over iterations up to allowed maximum.
do iter=1,mxit
    its=iter
    b=A(m+1)
    err=ABS(b)
    d=DCMPLX(0.d0,0.d0)
    f=DCMPLX(0.d0,0.d0)
    abx=ABS(x)
    do j=m,1,-1
    !           Efficient computation of the polynomial and its first TWO
    !           derivatives.
        f=x*f+d
        d=x*d+b
        b=x*b+A(j)
        err=ABS(b)+abx*err
10      continue
    END DO
    err=epss*err
    !
    if(ABS(b).le.err) then
    !           Special case: we are on the root.
       return
    else
    !           The generic case; use Laguerre's formula.
       g=d/b
       g2=g*g
       h=g2-2.d0*f/b
       sq=SQRT((m-1)*(m*h-g2))
       gp=g+sq
       gm=g-sq
       abp=ABS(gp)
       abm=ABS(gm)
       if(abp.lt.abm) gp=gm
           if (MAX(abp,abm).gt.0.d0) then
              dx=m/gp
           else
              dx=CEXP(CMPLX(LOG(1.d0+abx),DBLE(iter)))
           endif
       endif
       x1=x-dx
       !         Check if we've converged.
       if(x.eq.x1)return
       if (MOD(iter,mt).ne.0) then
           x=x1
       else
       !
           x=x-dx*FRAC(iter/mt)
       endif
20    continue
END DO
write(6,*) 'Too many iterations'
return
end
```

Función rc()

```
!**********************************************************************
double precision function rc(x,y)
!**********************************************************************
!     PURPOSE: Compute Carlson degenerate integral RC
!              R_C(x,y)=1/2 \int_0^\infty dt (t+x)^(-1/2) (t+y)^(-1)
!     ARGUMENTS: x,y
!     ROUTINES CALLED:  None.
!     ALGORITHM: Due to B.C. Carlson.
!     ACCURACY:  The parameter ERRTOL sets the desired accuracy.
!     REMARKS:
!     AUTHOR:  Press et al (1992)
!     DATE WRITTEN:  25 Mar 91.
!     REVISIONS:
!**********************************************************************
double precision x,y,errtol,tiny,sqrtny,big,tnbg,comp1,comp2,third,a1,c2, &
c3,c4
parameter (errtol=0.0012d0,tiny=1.69d-38,sqrtny=1.3d-19,big=3.d37, &
tnbg=TINY*BIG,comp1=2.236d0/SQRTNY,comp2=TNBG*TNBG/25.d0,third=1.d0/3.d0, &
a1=.3d0,c2=1.d0/7.d0,c3=.375d0,c4=9.d0/22.d0)
double precision alamb,ave,s,w,xt,yt

if(y.gt.0.)then
    xt=x
    yt=y
    w=1.d0
else
    xt=x-y
    yt=-y
    w=sqrt(x)/sqrt(xt)
endif
1 continue
  alamb=2.d0*sqrt(xt)*sqrt(yt)+yt
  xt=.25d0*(xt+alamb)
  yt=.25d0*(yt+alamb)
  ave=third*(xt+yt+yt)
  s=(yt-ave)/ave
if(abs(s).gt.errtol)goto 1
rc=w*(1.d0+s*s*(a1+s*(c2+s*(c3+s*c4))))/sqrt(ave)
return
end
```

Función rd()

```
!***************************************************************************
double precision function rd(x,y,z)
!***************************************************************************
!     PURPOSE: Compute Carlson degenerate integral RD
!              R_D(x,y,z)=3/2 \int_0^\infty dt (t+x)^(-1/2) (t+y)^(-1/2) (t+z)^(-3/2)
!     ARGUMENTS: x,y,z
!     ROUTINES CALLED:  None.
!     ALGORITHM: Due to B.C. Carlson.
!     ACCURACY:  The parameter ERRTOL sets the desired accuracy.
!     REMARKS:
!     AUTHOR:  Press et al (1992)
!     DATE WRITTEN:  25 Mar 91.
!     REVISIONS:
!***************************************************************************
double precision x,y,z,errtol,tiny,big,a1,c2,c3,c4,c5,c6
parameter (errtol=0.0015d0,tiny=1.d-25,big=4.5d21,a1=3.d0/14.d0,c2=1.d0/6.d0, &
c3=9.d0/22.d0,c4=3.d0/26.d0,c5=0.25d0*C3,c6=1.5d0*C4)
double precision alamb,ave,delx,dely,delz,ea,eb,ec,ed,ee,fac,sqrtx,sqrty, &
sqrtz,sum,xt,yt,zt
xt=x
yt=y
zt=z
sum=0.d0
fac=1.d0
1     continue
      sqrtx=sqrt(xt)
      sqrty=sqrt(yt)
      sqrtz=sqrt(zt)
      alamb=sqrtx*(sqrty+sqrtz)+sqrty*sqrtz
      sum=sum+fac/(sqrtz*(zt+alamb))
      fac=0.25d0*fac
      xt=.25d0*(xt+alamb)
      yt=.25d0*(yt+alamb)
      zt=.25d0*(zt+alamb)
      ave=.2d0*(xt+yt+3.d0*zt)
      delx=(ave-xt)/ave
      dely=(ave-yt)/ave
      delz=(ave-zt)/ave
if(max(abs(delx),abs(dely),abs(delz)).gt.errtol)goto 1
ea=delx*dely
eb=delz*delz
ec=ea-eb
ed=ea-6.d0*eb
ee=ed+ec+ec
rd=3.d0*sum+fac*(1.d0+ed*(-a1+c5*ed-c6*delz*ee)+delz*(c2*ee+delz*(-c3* &
ec+delz*c4*ea)))/(ave*sqrt(ave))
return
end
```

B.3. Caso de estudio 3: Prime Numbers

B.3.1. Código Fuente Inicial

```
program main

c*********************************************************************72
c
cc MAIN is the main program for PRIME_SERIAL_PRB.
c
c  Discussion:
c
c    PRIME_SERIAL_PRB tests the PRIME_SERIAL library.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    06 August 2009
c
c  Author:
c
c    John Burkardt
c
implicit none

integer n_factor
integer n_hi
integer n_lo

call timestamp ( )
write ( *, '(a)' ) ' '
write ( *, '(a)' ) 'PRIME_SERIAL_PRB'
write ( *, '(a)' ) '  FORTRAN77 version'
write ( *, '(a)' ) '  Test the PRIME_SERIAL library.'

n_lo = 1
n_hi = 131072
n_factor = 2

call prime_number_sweep ( n_lo, n_hi, n_factor )

n_lo = 5
n_hi = 500000
n_factor = 10

call prime_number_sweep ( n_lo, n_hi, n_factor )
c
c  Terminate.
c
write ( *, '(a)' ) ' '
write ( *, '(a)' ) 'PRIME_SERIAL_PRB'
write ( *, '(a)' ) '  Normal end of execution.'
write ( *, '(a)' ) ' '
call timestamp ( )

stop
end
subroutine prime_number_sweep ( n_lo, n_hi, n_factor )

c*********************************************************************72
c
cc PRIME_SERIAL_NUMBER_SWEEP does repeated calls to PRIME_SERIAL_NUMBER.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    06 August 2009
c
c  Author:
c
c    John Burkardt
c
c  Parameters:
c
c    Input, integer N_LO, the first value of N.
c
c    Input, integer N_HI, the last value of N.
c
```

```
c    Input, integer N_FACTOR, the factor by which to increase N after
c    each iteration.
c
implicit none

integer i
integer  n
integer  n_factor
integer n_hi
integer n_lo
integer primes
double precision time1
double precision time2

write ( *, '(a)' ) ' '
write ( *, '(a)' ) 'TEST01'
write ( *, '(a)' )
&  '  Call PRIME_SERIAL_NUMBER to count the primes from 1 to N.'
write ( *, '(a)' ) ' '
write ( *, '(a)' ) '         N        Pi          Time'
write ( *, '(a)' ) ' '

n = n_lo

10    continue

if ( n .le. n_hi ) then

call cpu_time ( time1 )

call prime_number ( n, primes )

call cpu_time ( time2 )

write ( *, '(2x,i8,2x,i8,g14.6)' ) n, primes, time2 - time1

n = n * n_factor

go to 10

end if

return
end

subroutine prime_number ( n, total )

c**********************************************************************72
c
cc PRIME_NUMBER returns the number of primes between 1 and N.
c
c  Discussion:
c
c    A naive algorithm is used.
c
c    Mathematica can return the number of primes less than or equal to N
c    by the command PrimePi[N].
c
c                N  PRIME_NUMBER
c
c                1           0
c               10           4
c              100          25
c            1,000         168
c           10,000       1,229
c          100,000       9,592
c        1,000,000      78,498
c       10,000,000     664,579
c      100,000,000   5,761,455
c    1,000,000,000  50,847,534
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    20 May 2009
c
c  Author:
c
c    John Burkardt
c
c  Parameters:
c
c    Input, integer N, the maximum number to check.
```

```
c
c    Output, integer TOTAL, the number of prime numbers up to N.
c
implicit none

integer i
integer j
integer n
integer prime
integer total

total = 0

do i = 2, n

prime = 1

do j = 2, i - 1
if ( mod ( i, j ) == 0 ) then
prime = 0
go to 10
end if
end do

10      continue

total = total + prime

end do

return
end
subroutine timestamp ( )

c**********************************************************************72
c
cc TIMESTAMP prints out the current YMDHMS date as a timestamp.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    12 January 2007
c
c  Author:
c
c    John Burkardt
c
c  Parameters:
c
c    None
c
implicit none

character * ( 8 ) ampm
integer d
character * ( 8 ) date
integer h
integer m
integer mm
character * ( 9 ) month(12)
integer n
integer s
character * ( 10 ) time
integer y

save month

data month /
&  'January  ', 'February ', 'March    ', 'April    ',
&  'May      ', 'June     ', 'July     ', 'August   ',
&  'September', 'October  ', 'November ', 'December ' /

call date_and_time ( date, time )

read ( date, '(i4,i2,i2)' ) y, m, d
read ( time, '(i2,i2,i2,1x,i3)' ) h, n, s, mm

if ( h .lt. 12 ) then
ampm = 'AM'
else if ( h .eq. 12 ) then
if ( n .eq. 0 .and. s .eq. 0 ) then
ampm = 'Noon'
```

```
else
ampm = 'PM'
end if
else
h = h - 12
if ( h .lt. 12 ) then
ampm = 'PM'
else if ( h .eq. 12 ) then
if ( n .eq. 0 .and. s .eq. 0 ) then
ampm = 'Midnight'
else
ampm = 'AM'
end if
end if
end if

write ( *,
&  '(i2,1x,a,1x,i4,2x,i2,a1,i2.2,a1,i2.2,a1,i3.3,1x,a)' )
&  d, month(m), y, h, ':', n, ':', s, '.', mm, ampm

return
end
```

B.3.2. Código Fuente Paralelizado por los Autores

```
program main

c**********************************************************************72
c
cc MAIN is the main program for PRIME_NUMBER_OPENMP.
c
c  Discussion:
c
c    This program calls a version of PRIME_NUMBER that includes
c    OpenMP directives for parallel processing.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    21 May 2009
c
c  Author:
c
c    John Burkardt
c
implicit none

include 'omp_lib.h'

integer i
integer j
integer n
integer n_factor
integer n_hi
integer n_lo
integer primes
double precision wtime

write ( *, '(a)' ) ' '
write ( *, '(a)' ) 'PRIME_NUMBER_OPENMP'
write ( *, '(a)' ) '  FORTRAN77/OpenMP version'

write ( *, '(a)' ) ' '
write ( *, '(a,i8)' )
&  '  Number of processors available = ', omp_get_num_procs ( )
write ( * ,'(a,i8)' )
&  '  Number of threads =              ', omp_get_max_threads ( )

n_lo = 1
n_hi = 131072
n_factor = 2

call prime_number_sweep_openmp ( n_lo, n_hi, n_factor )

n_lo = 5
n_hi = 500000
n_factor = 10

call prime_number_sweep_openmp ( n_lo, n_hi, n_factor )
c
```

```
c  Terminate.
c
write ( *, '(a)' ) ' '
write ( *, '(a)' ) 'PRIME_NUMBER_OPENMP'
write ( *, '(a)' ) '  Normal end of execution.'

stop
end
subroutine prime_number_sweep_openmp ( n_lo, n_hi, n_factor )

c**********************************************************************72
c
cc PRIME_NUMBER_SWEEP_OPENMP does repeated calls to PRIME_NUMBER.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    06 August 2009
c
c  Author:
c
c    John Burkardt
c
c  Parameters:
c
c    Input, integer N_LO, the first value of N.
c
c    Input, integer N_HI, the last value of N.
c
c    Input, integer N_FACTOR, the factor by which to increase N after
c    each iteration.
c
implicit none

include 'omp_lib.h'

integer i
integer n
integer n_factor
integer n_hi
integer n_lo
integer primes
double precision wtime

write ( *, '(a)' ) ' '
write ( *, '(a)' ) '         N        Pi          Time'
write ( *, '(a)' ) ' '

n = n_lo

10    continue

if ( n <= n_hi ) then

wtime = omp_get_wtime ( )

call prime_number ( n, primes )

wtime = omp_get_wtime ( ) - wtime

write ( *, '(2x,i8,2x,i8,g14.6)' ) n, primes, wtime

n = n * n_factor

go to 10

end if

return
end
subroutine prime_number ( n, total )

c**********************************************************************72
c
cc PRIME_NUMBER returns the number of primes between 1 and N.
c
c  Discussion:
c
c    A naive algorithm is used.
c
c    Mathematica can return the number of primes less than or equal to N
c    by the command PrimePi[N].
c
```

```
c                N  PRIME_NUMBER
c
c                1           0
c               10           4
c              100          25
c            1,000         168
c           10,000       1,229
c          100,000       9,592
c        1,000,000      78,498
c       10,000,000     664,579
c      100,000,000   5,761,455
c    1,000,000,000  50,847,534
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    21 May 2009
c
c  Author:
c
c    John Burkardt
c
c  Parameters:
c
c    Input, integer N, the maximum number to check.
c
c    Output, integer TOTAL, the number of prime numbers up to N.
c
implicit none

integer i
integer j
integer n
integer prime
integer total

total = 0

c$omp parallel
c$omp&  shared ( n )
c$omp&  private ( i, j, prime )

c$omp do reduction ( + : total )

do i = 2, n

prime = 1

do j = 2, i - 1
if ( mod ( i, j ) == 0 ) then
prime = 0
go to 10
end if
end do

10      continue

total = total + prime

end do

c$omp end do

c$omp end parallel

return
end
```

B.3.3. Perfilado con Gprof

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls   s/call   s/call  name
100.20     42.20    42.20       24     1.76     1.76  prime_number_
0.00     42.20     0.00        2     0.00    21.10  prime_number_sweep_
0.00     42.20     0.00        2     0.00     0.00  timestamp_
0.00     42.20     0.00        1     0.00    42.20  MAIN__

%         the percentage of the total running time of the
time       program used by this function.

...

Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.02% of 42.20 seconds

index % time    self  children    called     name
42.20    0.00      24/24          prime_number_sweep_ [2]
[1]    100.0   42.20    0.00      24         prime_number_ [1]
-----------------------------------------------
0.00   42.20       2/2           MAIN__ [3]
[2]    100.0    0.00   42.20       2         prime_number_sweep_ [2]
42.20    0.00      24/24          prime_number_ [1]
-----------------------------------------------
0.00   42.20       1/1           main [4]
[3]    100.0    0.00   42.20       1         MAIN__ [3]
0.00   42.20       2/2           prime_number_sweep_ [2]
0.00    0.00       2/2           timestamp_ [5]
-----------------------------------------------
<spontaneous>
[4]    100.0    0.00   42.20                 main [4]
0.00   42.20       1/1           MAIN__ [3]
-----------------------------------------------
0.00    0.00       2/2           MAIN__ [3]
[5]      0.0    0.00    0.00       2         timestamp_ [5]
-----------------------------------------------

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.
...
...
...
Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Index by function name

[3] MAIN__                  [2] prime_number_sweep_
[1] prime_number_           [5] timestamp_
```

B.3.4. Corrida Original

Se listan los resultados de la ejecución del programa paralelizado por los autores y ejecutado en diversas configuraciones de procesadores y theards.

```
PRIME_NUMBER_OPEN_MP
FORTRAN77/OpenMP version

Number of processors available =       16
Number of threads =                     1

N        Pi          Time

1         0  0.255620E-04
2         1  0.237487E-05
4         2  0.195624E-05
8         4  0.202609E-05
16        6  0.230502E-05
32       11  0.342214E-05
64       18  0.621611E-05
128      31  0.163428E-04
256      54  0.490299E-04
512      97  0.186130E-03
1024    172  0.581997E-03
2048    309  0.206371E-02
4096    564  0.751715E-02
8192   1028  0.272762E-01
16384  1900  0.872091E-01
32768  3512  0.248464
65536  6542  0.899972
131072 12251  3.32134

N        Pi          Time

5         3  0.167591E-05
50       15  0.335183E-05
500      95  0.102598E-03
5000    669  0.685001E-02
50000  5133  0.527939
500000 41538  43.2832

PRIME_NUMBER_OPEN_MP
Normal end of execution.
```

```
PRIME_NUMBER_OPEN_MP
FORTRAN77/OpenMP version

Number of processors available =       16
Number of threads =                     2

N        Pi          Time

1         0  0.466897E-03
2         1  0.338741E-04
4         2  0.318480E-04
8         4  0.319183E-04
16        6  0.320566E-04
32       11  0.324068E-04
64       18  0.378541E-04
128      31  0.421153E-04
256      54  0.672578E-04
512      97  0.154142E-03
1024    172  0.518231E-03
2048    309  0.159164E-02
4096    564  0.556721E-02
8192   1028  0.198530E-01
16384  1900  0.654250E-01
32768  3512  0.177339
65536  6542  0.664624
131072 12251  2.43721

N        Pi          Time

5         3  0.593662E-04
50       15  0.365977E-04
500      95  0.135075E-03
5000    669  0.502119E-02
50000  5133  0.386618
500000 41538  31.8959

PRIME_NUMBER_OPEN_MP
Normal end of execution.
```

```
PRIME_NUMBER_OPEN_MP
FORTRAN77/OpenMP version

Number of processors available =       16
Number of threads =                     4

N        Pi          Time

1         0  0.321695E-03
2         1  0.710296E-04
4         2  0.643949E-04
8         4  0.699819E-04
16        6  0.697733E-04
32       11  0.622990E-04
64       18  0.740332E-04
128      31  0.758488E-04
256      54  0.845091E-04
512      97  0.132492E-03
1024    172  0.304723E-03
2048    309  0.915286E-03
4096    564  0.309444E-02
8192   1028  0.108371E-01
16384  1900  0.395972E-01
32768  3512  0.146115
65536  6542  0.544473
131072 12251  2.05273

N        Pi          Time

5         3  0.139057E-03
50       15  0.776649E-04
500      95  0.228245E-03
5000    669  0.433959E-02
50000  5133  0.325564
500000 41538  26.8655

PRIME_NUMBER_OPEN_MP
Normal end of execution.
```

```
PRIME_NUMBER_OPEN_MP
FORTRAN77/OpenMP version

Number of processors available =       16
Number of threads =                     8

N        Pi          Time

1         0  0.558740E-03
2         1  0.975700E-04
4         2  0.105392E-03
8         4  0.121806E-03
16        6  0.113354E-03
32       11  0.113704E-03
64       18  0.268544E-03
128      31  0.119361E-03
256      54  0.136193E-03
512      97  0.222588E-03
1024    172  0.471506E-03
2048    309  0.129956E-02
4096    564  0.440008E-02
8192   1028  0.825839E-02
16384  1900  0.257384E-01
32768  3512  0.746405E-01
65536  6542  0.235087
131072 12251  0.889289

N        Pi          Time

5         3  0.233413E-03
50       15  0.124599E-03
500      95  0.204708E-03
5000    669  0.206915E-02
50000  5133  0.145426
500000 41538  11.4197

PRIME_NUMBER_OPEN_MP
Normal end of execution.
```

B.4. Caso de estudio 4: QUAD_SERIAL

B.4.1. Código Fuente Original

```
program main

c**********************************************************************72
c
cc MAIN is the main program for QUAD_SERIAL.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    12 January 2007
c
c  Author:
c
c    John Burkardt
c
double precision a
double precision b
double precision error
double precision exact
external f
double precision f
integer i
integer n
double precision total
double precision wtime
double precision wtime1
double precision wtime2
double precision x

a =  0.0
b = 10.0
n = 10000000
exact = 0.499363

call timestamp ( )
write ( *, * ) ' '
write ( *, * ) 'QUAD:'
write ( *, * ) '  FORTRAN77 version'
write ( *, '(a)' ) '  Estimate the integral of f(x) from A to B.'
write ( *, '(a)' ) '  f(x) = 50 / (pi * ( 2500 * x * x + 1 ) ).'
write ( *, '(a)' ) ' '
write ( *, '(a,g14.6)' ) '  A        = ', a
write ( *, '(a,g14.6)' ) '  B        = ', b
write ( *, '(a,i16)' ) '  N        = ', n
write ( *, '(a,g14.6)' ) '  Exact    = ', exact


call cpu_time ( wtime1 )

total = 0.0D+00
do i = 1, n
x = ( ( n - i ) * a + ( i - 1 ) * b ) / ( n - 1 )
total = total + f ( x )
```

```
end do

call cpu_time ( wtime2 )

total = ( b - a ) * total / dble ( n )
error = abs ( total - exact )
wtime = wtime2 - wtime1

write ( *, '(a)' ) ' '
write ( *, '(a,g14.6)' ) '  Estimate = ', total
write ( *, '(a,g14.6)' ) '  Error    = ', error
write ( *, '(a,g14.6)' ) '  Time     = ', wtime
c
c  Terminate.
c
write ( *, * ) ' '
write ( *, * ) 'QUAD_SERIAL:'
write ( *, * ) '  Normal end of execution.'
write ( *, * ) ' '
call timestamp ( )

stop
end
function f ( x )

c**********************************************************************72
c
cc F evaluates the function.
c
double precision f
double precision pi
double precision x

pi = 3.141592653589793D+00
f = 50.0D+00 / ( pi * ( 2500.0D+00 * x * x + 1.0D+00 ) )

return
end
subroutine timestamp ( )

c**********************************************************************72
c
cc TIMESTAMP prints out the current YMDHMS date as a timestamp.
c
c  Licensing:
c
c    This code is distributed under the GNU LGPL license.
c
c  Modified:
c
c    12 January 2007
c
c  Author:
c
c    John Burkardt
c
c  Parameters:
c
c    None
c
implicit none
```

```
character * ( 8 ) ampm
integer d
character * ( 8 ) date
integer h
integer m
integer mm
character * ( 9 ) month(12)
integer n
integer s
character * ( 10 ) time
integer y

save month

data month /
&  'January  ', 'February ', 'March    ', 'April    ',
&  'May      ', 'June     ', 'July     ', 'August   ',
&  'September', 'October  ', 'November ', 'December ' /

call date_and_time ( date, time )

read ( date, '(i4,i2,i2)' ) y, m, d
read ( time, '(i2,i2,i2,1x,i3)' ) h, n, s, mm

if ( h .lt. 12 ) then
ampm = 'AM'
else if ( h .eq. 12 ) then
if ( n .eq. 0 .and. s .eq. 0 ) then
ampm = 'Noon'
else
ampm = 'PM'
end if
else
h = h - 12
if ( h .lt. 12 ) then
ampm = 'PM'
else if ( h .eq. 12 ) then
if ( n .eq. 0 .and. s .eq. 0 ) then
ampm = 'Midnight'
else
ampm = 'AM'
end if
end if
end if

write ( *,
&  '(i2,1x,a,1x,i4,2x,i2,a1,i2.2,a1,i2.2,a1,i3.3,1x,a)' )
&  d, month(m), y, h, ':', n, ':', s, '.', mm, ampm

return
end
```

B.4.2. Perfilado del Programa QUAD_SERIAL: Ejecución Secuencial

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
65.51      0.17     0.17 10000000     0.00     0.00  timestamp_
26.98      0.24     0.07                             MAIN__
7.71      0.26     0.02        1    20.04    20.04  f_
0.00      0.26     0.00        2     0.00     0.00  register_tm_clones

%         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
seconds   for by this function and those listed above it.

self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
listing.

calls      the number of times this function was invoked, if
this function is profiled, else blank.

self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
else blank.

total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
function is profiled, else blank.

name       the name of the function.  This is the minor sort
for this listing. The index shows the location of
the function in the gprof listing. If the index is
in parenthesis it shows where it would appear in
the gprof listing if it were to be printed.


Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 3.84% of 0.26 seconds

index % time    self  children    called     name
<spontaneous>
[1]     92.3    0.07    0.17                 MAIN__ [1]
0.17    0.00 10000000/10000000     timestamp_ [2]
0.00    0.00       1/2           register_tm_clones [5]
-----------------------------------------------
0.17    0.00 10000000/10000000     MAIN__ [1]
```

```
[2]     65.4    0.17    0.00 10000000          timestamp_ [2]
-----------------------------------------------
0.02    0.00       1/1           main [4]
[3]      7.7    0.02    0.00       1          f_ [3]
0.00    0.00       1/2           register_tm_clones [5]
-----------------------------------------------
<spontaneous>
[4]      7.7    0.00    0.02                  main [4]
0.02    0.00       1/1           f_ [3]
-----------------------------------------------
0.00    0.00       1/2           f_ [3]
0.00    0.00       1/2           MAIN__ [1]
[5]      0.0    0.00    0.00       2          register_tm_clones [5]
-----------------------------------------------

This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.
...
...
...

Copyright (C) 2012 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.


Index by function name

[1] MAIN__ (quad_serial.f90) [5] register_tm_clones
[3] f_                       [2] timestamp_
```

B.4.3. Salida de la ejecución del Programa Original QUAD_SERIAL

```
14 December  2011   8:28:12.640 AM

QUAD:
FORTRAN77 version
Estimate the integral of f(x) from A to B.
f(x) = 50 / (pi * ( 2500 * x * x + 1 ) ).

A        =    0.00000
B        =    10.0000
N        =    10000000
Exact    =    0.499363

Estimate =   0.499371
Error    =   0.790784E-05
Time     =   0.338820

QUAD_SERIAL:
Normal end of execution.

14 December  2011   8:28:12.980 AM
```

B.4.4. Salida de la ejecución al Inicio del Proceso

```
16 July      2015   6:25:48.719 PM

QUAD:
FORTRAN77 version
Estimate the integral of f(x) from A to B.
f(x) = 50 / (pi * ( 2500 * x * x + 1 ) ).

A        =    0.00000
B        =    10.0000
N        =    10000000
Exact    =    0.499363

Estimate =   0.499371
Error    =   0.790784E-05
Time     =   0.308151

QUAD_SERIAL:
Normal end of execution.

16 July      2015   6:25:49.027 PM
```

B.4.5. Ejecución de la iteración previa a la paralelización

```
16 July      2015   6:40:45.937 PM

QUAD:
FORTRAN77 version
Estimate the integral of f(x) from A to B.
f(x) = 50 / (pi * ( 2500 * x * x + 1 ) ).

A        =    0.00000
B        =    10.0000
N        =    10000000
Exact    =    0.499363

Estimate =   0.499371
Error    =   0.790784E-05
Time     =   0.320313

QUAD_SERIAL:
Normal end of execution.

16 July      2015   6:40:46.257 PM
```

B.4.6. Salida de la Ejecución Tras la Paralelización

```
16 July      2015   7:15:37.080 PM

QUAD:
FORTRAN77 version
Estimate the integral of f(x) from A to B.
f(x) = 50 / (pi * ( 2500 * x * x + 1 ) ).

A        =    0.00000
B        =    10.0000
N        =    10000000
Exact    =    0.499363

Estimate =   0.499371
Error    =   0.790784E-05
Time     =   0.102887

QUAD_SERIAL:
Normal end of execution.

16 July      2015   7:15:37.183 PM
```

Apéndice C

Generador de árboles para latex

A continuación se muestra la implementación trivial del patrón visitante para generar el árbol de sintaxis abstracta según lo requiere el paquete Tikz de latex. Esta implementación se realizó para demostrar la facilidad con la cual puede utilizarse la infraestructura de Photran para el análisis de código fuente escrito en Fortran. La Figura C.1 muestra el código java necesario para dibujar, según lo requiere el paquete tikz de latex, el AST de un programa Fortran, cabe destacar que realizar dicha tarea a mano es muy compleja.

```
public static class AstToTex {

        final static String lBracket="[ ";
        final static String rBracket=" ]";
        final static String dot=".";

        public static String AstToTex(IFortranAST ast){

        String str="\\Tree"+lBracket + dot ;
        IASTNode node= ast.getRoot();
        str=str.concat(ast.getClass().getSimpleName());
        str+=Nodes(node);
        str=str + rBracket;
        return str;
}


private static String Nodes(IASTNode node) {
        Iterator<? extends IASTNode> children = node.getChildren().iterator();
        String Str="";
        while( children.hasNext()){
                Str+=lBracket + dot ;
                IASTNode child=children.next();
                String methosDesc=addMethodDescriptions(child);

                if (child instanceof Token) {
                        String tokenStr=((Token)child).getText();
                                if ( tokenStr.equals("\n") )
                                        tokenStr= "{new line}" ;
                                Str=Str+ tokenStr ;
                }
                else{
                        Str=Str +  child.getClass().getSimpleName() + " " ;
                }
                Str=Str.concat(Nodes(child)+ rBracket);
        }
        return Str;
}

}
```

Figura C.1: Clase java implementando el patrón visitor para generar el comando latex para la impresión del AST de cualquier programa Fortran

Bibliografía

[1] Photran, an Integrated Development Environment and Refactoring Tool for Fortran. http://www.eclipse.org/photran/. [cited at p. 74]

[2] Fortran anecdotes. *IEEE Annals of the History of Computing*, 6(1):59–64, 1984. [cited at p. 144]

[3] Datamonitor cobol – continuing to drive value in the 21st century. IEEE, 2008. [cited at p. 38]

[4] Openmp architecture review board, "openmp application program interface - version 3.1", July 2011. [cited at p. 121, 232]

[5] V. Agarwal, MS Hrishikesh, S.W. Keckler, and D. Burger. Clock rate versus ipc: The end of the road for conventional microarchitectures. *ACM SIGARCH Computer Architecture News*, 28(2):248–259, 2000. [cited at p. 32, 33]

[6] Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: principles, techniques, & tools*, volume 1009. Pearson/Addison Wesley, 2007. [cited at p. 8]

[7] Albert Alderson and Hanifa Shah. Technical opinion: Viewpoints on legacy systems. *Communications of the ACM*, 42(3):115–116, 1999. [cited at p. 5]

[8] American National Standards Institute. X3. 9-1978. *American National Standards Institute, New York*, 1978. [cited at p. 72, 195]

[9] American National Standards Institute and Computer and Business Equipment Manufacturers Association. *American National Standard for programming language, FORTRAN — extended: ANSI X3.198-1992: ISO/IEC 1539: 1991 (E)*. American National Standards Institute, September 1992. [cited at p. 4, 92, 111, 117, 157, 180, 195, 204]

[10] RS Arnold. Software restructuring. *Proceedings of the IEEE*, 77(4):607–617, 1989. [cited at p. 39, 40]

[11] Lerina Aversano, Luigi Cerulo, and Ciro Palumbo. Mining candidate web services from legacy code. In *Web Site Evolution, 2008. WSE 2008. 10th International Symposium on*, pages 37–40. IEEE, 2008. [cited at p. 30]

[12] J. Backus. The history of Fortran I, II, and III. *ACM SIGPLAN Notices*, 13(8):165–180, 1978. [cited at p. 3, 18]

[13] J. Backus. The History of Fortran I, II, and III. *ACM SIGPLAN Notices*, 13(8):165–180, 1978. [cited at p. 17, 19, 95, 144]

[14] JW Backus, RJ Beeber, S. Best, R. Goldberg, LM Haibt, HL Herrick, RA Nelson, D. Sayre, PB Sheridan, H. Stern, et al. The FORTRAN Automatic Coding System. In *Papers presented at the February 26-28, 1957, western joint computer conference: Techniques for reliability*, pages 188–198. ACM, 1957. [cited at p. 3, 18]

[15] Brenda S Baker. An algorithm for structuring programs. In *Proceedings of the 3rd ACM SIGACT-SIGPLAN symposium on Principles on programming languages*, pages 113–126. ACM, 1976. [cited at p. 41]

[16] Brenda S Baker. An algorithm for structuring flowgraphs. *Journal of the ACM (JACM)*, 24(1):98–120, 1977. [cited at p. 41]

[17] David W Balmer and Ray J Paul. Casm-the right environment for simulation. *Journal of the Operational Research Society*, pages 443–452, 1986. [cited at p. 17]

[18] Gary Barnett. The future of the mainframe. *Ovum, october*, 2005. [cited at p. 38]

[19] Pablo Barrio, Carlos Carreras, Roberto Sierra, Tobias Kenter, and Christian Plessl. Turning control flow graphs into function calls: Code generation for heterogeneous architectures. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 559–565. IEEE, 2012. [cited at p. 17]

[20] Ilija Basicevic, Sabina Jovanovic, Branislav Drapsin, Miroslav Popovic, and Vladislav Vrtunski. An approach to parallelization of legacy software. In *Engineering of Computer Based Systems, 2009. ECBS-EERC'09. First IEEE Eastern European Conference on the*, pages 42–48. IEEE, 2009. [cited at p. 41]

[21] Victor R Basili, Daniela Cruzes, Jeffrey C Carver, Lorin M Hochstein, Jeffrey K Hollingsworth, Marvin V Zelkowitz, and Forrest Shull. Understanding the high-performance-computing community. 2008. [cited at p. 3, 7, 17, 30, 31, 34]

[22] Victor R. Basili, Richard W Selby Jr, and T Phillips. Metric analysis and data validation across fortran projects. *Software Engineering, IEEE Transactions on*, (6):652–663, 1983. [cited at p. 84, 86, 88]

[23] Victor R. Basili and Albert J. Turner. Iterative enhancement: A practical technique for software development. *Software Engineering, IEEE Transactions on*, (4):390–396, 1975. [cited at p. 62]

[24] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mallor, Ken Shwaber, and Jeff Sutherland. The Agile Manifesto. Technical report, The Agile Alliance, 2001. [cited at p. 59]

[25] Keith Bennett. Legacy systems: coping with stress. *Software, IEEE*, 12(1):19–23, 1995. [cited at p. 5, 29]

[26] Keith H Bennett and Václav T Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM, 2000. [cited at p. 5]

[27] Arthur J Bernstein. Analysis of programs for parallel processing. *Electronic Computers, IEEE Transactions on*, (5):757–763, 1966. [cited at p. 121]

[28] M Berz. High-order computation and normal form analysis of repetitive systems. *Physics of Particle Accelerators, volume AIP*, 249:456, 1991. [cited at p. 3, 34]

[29] Barry Boehm and Victor R Basili. Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili*, 426, 2005. [cited at p. 37]

[30] Barry W Boehm. The high cost of software. *Practical Strategies for Developing Large Software Systems*, pages 3–15, 1975. [cited at p. 37]

[31] Corrado Böhm and Giuseppe Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5):366–371, 1966. [cited at p. 39]

[32] Grady Booch. The complexity of programming models. *Keynote talk at AOSD*, pages 14–18, 2005. [cited at p. 38]

[33] Ghizlane El Boussaidi, Alvine Boaye Belle, Stephane Vaucher, and Hafedh Mili. Reconstructing architectural views from legacy systems. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 345–354. IEEE, 2012. [cited at p. 29]

[34] G Bracchi and D Ferrari. A graphic language for describing and manipulating two-dimensional patterns. In *Advanced Computer Graphics*, pages 263–277. Springer, 1971. [cited at p. 3, 34]

[35] M.L. Brodie and M. Stonebraker. *Migrating legacy systems.* Morgan Kaufmann Publishers, 1995. [cited at p. 25]

[36] Carole Brooke and Magnus Ramage. Organisational scenarios and legacy systems. *International Journal of Information Management*, 21(5):365–384, 2001. [cited at p. 27]

[37] FP Brooks. The mythical man-month, 1975. [cited at p. 89]

[38] F.P. Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE computer*, 20(4):10–19, 1987. [cited at p. 36, 40, 59, 60, 69, 84]

[39] Shirley Browne, Jack Dongarra, Nathan Garner, George Ho, and Philip Mucci. A portable programming interface for performance evaluation on modern processors. *International Journal of High Performance Computing Applications*, 14(3):189–204, 2000. [cited at p. 127]

[40] Maarten Bullynck and Liesbeth De Mol. Setting-up early computer programs: Dh lehmer's eniac computation. *Archive for Mathematical Logic*, 49(2):123–146, 2010. [cited at p. 13]

[41] Necdet Bulut, Maurice H Halstead, and Rudolf Bayer. Experimental validation of a structural property of fortran algorithms. In *Proceedings of the 1974 annual conference-Volume 1*, pages 207–211. ACM, 1974. [cited at p. 88]

[42] Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, and Porfirio Tramontana. Migrating interactive legacy systems to web services. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 10–pp. IEEE, 2006. [cited at p. 30]

[43] Richard G Canning. That maintenance 'iceberg'. *EDP Analyzer*, 10(10):1–14, 1972. [cited at p. 35]

[44] Jeffrey C Carver, L Hochstein, Richard P Kendall, Taiga Nakamura, Marvin V Zelkowitz, Victor R Basili, and Douglass E Post. Observations about software development for high end computing. *CTWatch Quarterly*, 2(4A):33–37, 2006. [cited at p. 3, 17, 31, 34]

[45] Jeffrey C Carver, Richard P Kendall, Susan E Squires, and Douglass E Post. Software development environments for scientific and engineering software: A series of case studies. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 550–559. IEEE, 2007. [cited at p. 31]

[46] N. Chapin, J.E. Hale, K.M. Khan, J.F. Ramil, and W.G. Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(1), 2001. [cited at p. 36]

[47] E Cheney and David Kincaid. *Numerical mathematics and computing*. Cengage Learning, 1985. [cited at p. 141, 222]

[48] E Cheney and David Kincaid. *Numerical mathematics and computing*. Cengage Learning, 1985. [cited at p. 141]

[49] E.J. Chikofsky and J.H. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17, 1990. [cited at p. 39, 40]

[50] Ian Chivers and Jane Sleightholme. Compiler support for the fortran 2003 and 2008 standards revision 15. In *ACM SIGPLAN Fortran Forum*, volume 33, pages 38–51. ACM, 2014. [cited at p. 24]

[51] Maria Wahid Chowdhury and Muhammad Zafar Iqbal. Integration of legacy systems in software architecture. *SAVCBS 2004 Specification and Verification of Component-Based Systems*, page 110, 2004. [cited at p. 29]

[52] Eric Clayberg and Dan Rubel. *Eclipse: building commercial-quality plug-ins*, volume 2. Addison-Wesley Reading, 2004. [cited at p. 128]

[53] M. Cohen. Fortran: A few historical details. http://www.nag.co.uk/nagware/np/doc/fhistory.asp, Oct. 2004. [cited at p. 23]

[54] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994. [cited at p. 8, 89]

[55] Massimo Colosimo, Andrea De Lucia, Giuseppe Scanniello, and Genoveffa Tortora. Evaluating legacy system migration technologies through empirical studies. *Information and Software Technology*, 51(2):433–447, 2009. [cited at p. 29]

[56] International Business Machines Corporation. *Reference manual [S]: FORTRAN II for the IBM 704 data processing system.* 1958. [cited at p. 19]

[57] Frank P. Coyle. Legacy integration-changing perspectives. *IEEE Softw.*, 17(2):37–41, March 2000. [cited at p. 38]

[58] Dermott E Cullen. Endf/x: an extended endf format (evolution, not revolution). 2012. [cited at p. 3, 34]

[59] Marco D'Ambros. Supporting software evolution analysis with historical dependencies and defect information. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 412–415. IEEE, 2008. [cited at p. 37]

[60] Guy de Balbine. Better manpower utilization using automatic restructuring. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, AFIPS '75, pages 319–327, New York, NY, USA, 1975. ACM. [cited at p. 39, 41]

[61] Andrea De Lucia, Rita Francese, Giuseppe Scanniello, and Genoveffa Tortora. Developing legacy system migration methods and tools for technology transfer. *Software: Practice and Experience*, 38(13):1333–1364, 2008. [cited at p. 29]

[62] LS Dederick. The mathematics of exterior ballistic computations. *American Mathematical Monthly*, pages 628–634, 1940. [cited at p. 13]

[63] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-oriented reengineering patterns.* Elsevier, 2002. [cited at p. 26]

[64] Jason Dexter and Eric Agol. Geokerr: A fast new public code for computing photon orbits in a kerr spacetime. *Astrophysics Source Code Library*, 1:11015, 2010. [cited at p. 225, 232, 259]

[65] Erik H D'Hollander, Fubo Zhang, and Qi Wang. The fortran parallel transformer and its programming environment. *Information sciences*, 106(3):293–317, 1998. [cited at p. 41]

[66] GHF Diercksen, NE Grüner, and J Steuerwald. Computers and computation in molecular physics. In *Methods in Computational Molecular Physics*, pages 335–350. Springer, 1983. [cited at p. 3, 34]

[67] E.W. Dijkstra. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968. [cited at p. 39]

[68] Frederick J Dillman, Edward S Ferrara, and Sumeet Malhotra. System and method for using blueprints to provide a software solution for an enterprise, June 9 2009. US Patent 7,546,575. [cited at p. 29]

[69] Gleison S do Nascimento, Cirano Iochpe, Lucinéia Thom, André C Kalsing, and Álvaro Moreira. Identifying business rules to legacy systems reengineering based on bpm and soa. In *Computational Science and Its Applications–ICCSA 2012*, pages 67–82. Springer, 2012. [cited at p. 29]

[70] Jack Dongarra, Dennis Gannon, Geoffrey Fox, and Ken Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1):1–10, 2007. [cited at p. 3]

[71] Steve M Easterbrook and Timothy C Johns. Engineering the software for understanding climate change. *Computing in Science & Engineering*, 11(6):65–74, 2009. [cited at p. 30, 31]

[72] A. Eastwood. Firm fires shots at legacy systems. *Computing Canada*, 19(2):17, 1993. [cited at p. 89]

[73] Alison Eastwood. Firm fires shots at legacy systems. *Computing Canada*, 19(2):17, 1993. [cited at p. 37]

[74] Paul N Edwards. A brief history of atmospheric general circulation modeling. *International Geophysics*, 70:67–90, 2001. [cited at p. 13, 14, 15, 31]

[75] Rudolf Eigenmann, Jay Hoeflinger, Greg Jaxon, Zhiyuan Li, and David Padua. Restructuring fortran programs for cedar. In *IN PROCEEDINGS OF THE 1991 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING*, pages 57–66, 1991. [cited at p. 41]

[76] Mohammad El-Ramly, Paul Iglinski, Eleni Stroulia, Paul Sorenson, and Bruce Matichuk. Modeling the system-user dialog using interaction traces. In *Reverse Engi-*

neering, 2001. Proceedings. Eighth Working Conference on, pages 208–217. IEEE, 2001. [cited at p. 29]

[77] L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23, 2000. [cited at p. 5, 7, 89]

[78] Len Erlikh. Leveraging legacy system dollars for e-business. *IT professional*, 2(3):17–23, 2000. [cited at p. 37, 38]

[79] Michael Feathers. *Working effectively with legacy code*. Prentice Hall Professional, 2004. [cited at p. 26]

[80] Louis Fein. The role of the university in computers, data processing, and related fields. In *Managing Requirements Knowledge, International Workshop on*, pages 119–119. IEEE Computer Society, 1899. [cited at p. 3]

[81] Louis Fein. The role of the university in computers, data processing, and related fields. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 119–126. ACM, 1959. [cited at p. 30]

[82] B. Foote and J. Yoder. Big ball of mud. *Pattern languages of program design*, 4(654-692):99, 2000. [cited at p. 54]

[83] John Foreman, David A Fisher, Michael Bray, Kimberly Brune, and Mark Gerken. C4 software technology reference guide-a prototype, 1997. [cited at p. 86]

[84] A. FORTRAN. X3. 9-1978. *American National Standards Institute, New York*, 1978. [cited at p. 4, 21, 72, 144]

[85] A. FORTRAN. X3.198-1992. *American National Standards Institute, New York*, 1992. [cited at p. 4, 22, 118, 144]

[86] ANSI FORTRAN. X3. 9-1966. *American National Standards Institute Incorporated, New York*, page 40, 1966. [cited at p. 17, 144]

[87] A.S. FORTRAN. X3. 9-1966. *American National Standards Institute Incorporated, New York*, 1966. [cited at p. 4, 19, 144, 171, 195]

[88] The Eclipse Foundation. Eclipse.org home. http://www.eclipse.org/, 2010. [cited at p. 74]

[89] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999. [cited at p. 40]

[90] W Barkley Fritz. Eniac-a problem solver. *Annals of the History of Computing, IEEE*, 16(1):25–45, 1994. [cited at p. 3, 14]

[91] Maulahikmah Galinium and Negar Shahbaz. Success factors model: Case studies in the migration of legacy systems to service oriented architecture. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 236–241. IEEE, 2012. [cited at p. 30]

[92] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Reading, MA, 1995. [cited at p. 78, 93, 97]

[93] Erich Gamma and Kent Beck. *Contributing to Eclipse: principles, patterns, and plug-ins*. Addison-Wesley Professional, 2004. [cited at p. 128]

[94] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005. [cited at p. 33]

[95] Jeremy Gibbons. Fission for program comprehension. In *Mathematics of Program Construction*, pages 162–179. Springer, 2006. [cited at p. 29]

[96] Nicolas Gold. *The Meaning of"legacy Systems*

. Department of Computer Science, University of Durham, 1998. [cited at p. 5, 25]

[97] Herman H Goldstine. *The computer from Pascal to von Neumann*. Princeton University Press, 1972. [cited at p. 14]

[98] Julian E Gomez. An interactive fortran structuring aid. In *Proceedings of the 4th international conference on Software engineering*, pages 241–244. IEEE Press, 1979. [cited at p. 41]

[99] Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen, and Zhenghu Gong. The characteristics of cloud computing. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 275–279. IEEE, 2010. [cited at p. 5, 7]

[100] N Gorla. Techniques for application software maintenance. *Information and Software Technology*, 33(1):65–73, 1991. [cited at p. 37]

[101] William G Griswold and David Notkin. Automated assistance for program restructuring. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2(3):228–269, 1993. [cited at p. 40]

[102] He Guo, Chunyan Guo, Feng Chen, and Hongji Yang. Wrapping client-server application to web services for internet computing. In *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, pages 366–370. IEEE, 2005. [cited at p. 30]

[103] Maurice Halstead and Rudolf Bayer. Algorithm dynamics. In *Proceedings of the ACM annual conference*, pages 126–135. ACM, 1973. [cited at p. 8, 86, 87]

[104] Maurice H Halstead. Natural laws controlling algorithm structure? *ACM Sigplan Notices*, 7(2):19–26, 1972. [cited at p. 8, 86, 87, 88]

[105] Maurice H Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977. [cited at p. 8, 86, 87]

[106] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. IEEE Computer Society, 2009. [cited at p. 31]

[107] Don Heller. A survey of parallel algorithms in numerical linear algebra. *Siam Review*, 20(4):740–777, 1978. [cited at p. 6, 121]

[108] Gideon Hollander, Iddo Levinson, Benjamin Schlesinger, Guy Sheffer, and Avner Zangvil. System and method for developing new services from legacy computer applications, May 17 2004. US Patent App. 10/846,924. [cited at p. 29]

[109] Ellis Horowitz. Fortran can it be structured-should it be? *Computer*, 8(6):30–37, 1975. [cited at p. 41]

[110] S Huff. Information systems maintenance. *The Business Quarterly*, 55(1):30–32, 1990. [cited at p. 37]

[111] Sascha Hunold, Björn Krellner, Thomas Rauber, Thomas Reichel, and Gudula Rünger. Pattern-based refactoring of legacy software systems. In *Enterprise Information Systems*, pages 78–89. Springer, 2009. [cited at p. 29]

[112] IEEE. *IEEE Standard for Software Maintenance, IEEE Std 1219-1998*, volume 2. IEEE Press, 1999. [cited at p. 35]

[113] Intel. *Pentium Processor Family Developer's Manual Volume 3: Architecture and Programming Manual*. Mt. Prospect, IL, USA., 1995. [cited at p. 125]

[114] Intel Intel. Intel 64 and ia-32 architectures software developer's manual. *Volume 3ª*, 2013. [cited at p. 125]

[115] ISO. ANSI/ISO/IEC 1539-1:1997: Information technology — programming languages — Fortran — part 1: Base language. [cited at p. 23]

[116] ISO. ANSI/ISO/IEC 1539-1:2004(E): Information technology — Programming languages — Fortran Part 1: Base Language. pages xiv + 569, May 2004. [cited at p. 4, 24, 111, 144]

[117] Iso. International Standard - ISO/IEC 14764 IEEE Std 14764-2006. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, pages 1–46, 2006. [cited at p. 36]

[118] ISO. *ISO/IEC 1539-1:2010 Information technology — Programming languages — Fortran — Part 1: Base language*. International Organization for Standardization, Geneva, Switzerland, June 2010. [cited at p. 4, 24]

[119] Ralph E Johnson. Software development is program transformation. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 177–180. ACM, 2010. [cited at p. 35, 311]

[120] Alan H Karp. Programming for parallelism. *Computer*, 20(5):43–57, 1987. [cited at p. 3, 17, 34]

[121] Diane F Kelly. A software chasm: Software engineering and scientific computing. *Software, IEEE*, 24(6):120–119, 2007. [cited at p. 7, 30, 31]

[122] Richard Kendall, Jeffrey C Carver, David Fisher, Dale Henderson, Andrew Mark, Douglass Post, Clifford E Rhoades, and Susan Squires. Development of a weather forecasting code: A case study. *Software, IEEE*, 25(4):59–65, 2008. [cited at p. 31]

[123] Roy Kerr. Gravitational field of a spinning mass as an example of algebraically special metrics. *Physical review letters*, (11):237–238, 1963. [cited at p. 226]

[124] Kostas Kontogiannis and Prashant Patil. Evidence driven object identification in procedural code. In *Software Technology and Engineering Practice, 1999. STEP'99. Proceedings*, pages 12–21. IEEE, 1999. [cited at p. 38]

[125] Kostas A Kontogiannis, Renator DeMori, Ettore Merlo, Michael Galler, and Morris Bernstein. Pattern matching for clone and concept detection. In *Reverse engineering*, pages 77–108. Springer, 1996. [cited at p. 29]

[126] Arun Lakhotia and J-C Deprez. Restructuring functions with low cohesion. In *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on*, pages 36–46. IEEE, 1999. [cited at p. 26]

[127] Monica Lam, Ravi Sethi, JD Ullman, and Alfred Aho. Compilers: Principles, techniques, and tools, 2006. [cited at p. 75]

[128] Ralf Lammel and Chris Verhoef. Cracking the 500-language problem. *Software, IEEE*, 18(6):78–88, 2001. [cited at p. 38]

[129] Craig Larman and Victor R Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, 2003. [cited at p. 61, 69]

[130] Eunjoo Lee, Byungjeong Lee, Woochang Shin, and Chisu Wu. A reengineering process for migrating from an object-oriented legacy system to a component-based system. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 336–341. IEEE, 2003. [cited at p. 5]

[131] Gyungho Lee, Clyde P. Kruskal, and David J Kuck. An empirical study of automatic restructuring of nonnumerical programs for parallel processors. *Computers, IEEE Transactions on*, 100(10):927–933, 1985. [cited at p. 41]

[132] Jonathan Lee, Shin-Jie Lee, and Shang-Pin Ma. Deliver advanced traveler information services. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 682–687. IEEE, 2005. [cited at p. 26]

[133] Meir M Lehman. *Laws of program evolution-rules and tools for programming management*. 1978. [cited at p. 36, 54]

[134] Meir M Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221, 1980. [cited at p. 36, 54]

[135] Meir M Lehman, Juan F Ramil, Paul D Wernick, Dewayne E Perry, and Wladyslaw M Turski. Metrics and laws of software evolution-the nineties view. In *Software Metrics Symposium, 1997. Proceedings., Fourth International*, pages 20–32. IEEE, 1997. [cited at p. 36, 54]

[136] M.M. Lehman et al. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980. [cited at p. 54]

[137] M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski. Metrics and laws of software evolution-the nineties view. In *4th International Software Metrics Symposium (METRICS'97)*, 1997. [cited at p. 54]

[138] Xiangyu Li. A multi-agent based legacy information system integration strategy. In *Networking and Digital Society (ICNDS), 2010 2nd International Conference on*, volume 2, pages 72–75. IEEE, 2010. [cited at p. 30]

[139] Yang Li and Hongji Yang. Simplicity: a key engineering concept for program understanding. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 98–107. IEEE, 2001. [cited at p. 26]

[140] Bennet P Lientz, E. Burton Swanson, and Gail E Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978. [cited at p. 37]

[141] Yan Liu, Qingling Wang, Mingguang Zhuang, and Yunyun Zhu. Reengineering legacy systems with restful web service. In *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, pages 785–790. IEEE, 2008. [cited at p. 30]

[142] Kevin London, Shirley Moore, Philip Mucci, Keith Seymour, and Richard Luczak. The papi cross-platform interface to hardware performance counters. In *Department of Defense Users Group Conference Proceedings*, pages 18–21, 2001. [cited at p. 127]

[143] Peter Lynch. The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, 227(7):3431–3444, 2008. [cited at p. 13, 14, 15, 16, 31]

[144] Michael S Mahoney. What makes the history of software hard. *IEEE Annals of the History of Computing*, 1(3):8–18, 2008. [cited at p. 25]

[145] Terje Mathisen. Pentium secrets. *Byte magagize*, 1994. [cited at p. 125]

[146] Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976. [cited at p. 8, 84, 86, 87]

[147] James R McKee. Maintenance as a function of design. In *Proceedings of the July 9-12, 1984, national computer conference and exposition*, pages 187–193. ACM, 1984. [cited at p. 37]

[148] M. Méndez, A. Garrido, J. Overbey, F.G. Tinetti, and R. Johnson. Refactorización en Código Fortran Heredado. [cited at p. 181, 311]

[149] M. Mendez, J. Overbey, A. Garrido, F. Tinetti, and R. Johnson. A catalog and classification of fortran refactorings. In *11th Argentine Symposium on Software Engineering (ASSE 2010*, pages 1–10, 2010. [cited at p. 41, 103, 181, 311]

[150] M. Méndez, J. Overbey, A. Garrido, F. Tinetti, and R. Johnson. A Catalog and Two Possible Classifications of Fortran Refactorings. *Technical Report*, 2010. [cited at p. 103, 181]

[151] Mariano Méndez. *Fortran refactoring for legacy systems*. PhD thesis, Facultad de Informática, 2011. [cited at p. 17, 251]

[152] Mariano Méndez, Jeffrey Overbey, and Fernando Gustavo Tinetti. Legacy fortran software: Applying syntactic metrics to global climate models. In *XVIII Congreso Argentino de Ciencias de la Computación*, 2012. [cited at p. 85, 312]

[153] Mariano Méndez, Fernando G Tinetti, and Jeffrey L Overbey. Climate models: challenges for fortran development tools. In *Proceedings of the 2nd International workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, pages 6–12. IEEE Press, 2014. [cited at p. 85, 312]

[154] Mariano Méndez and Fernando Gustavo Tinetti. First steps towards a tool for legacy systems. In *XVII Congreso Argentino de Ciencias de la Computación*, 2011. [cited at p. 311]

[155] Mariano Méndez and Fernando Gustavo Tinetti. Aplicaciones científicas numéricas: El (ciclo de vida del) software heredado. In *XIV Workshop de Investigadores en Ciencias de la Computación*, 2012. [cited at p. 312]

[156] T. Mens and T. Tourwé. A survey of software refactoring. *IEEE Transactions on software engineering*, 30(2):126–139, 2004. [cited at p. 40]

[157] M. Metcalf. The seven ages of fortran. *Journal of Computer Science and Technology*, 11(1):1–8, 2011. [cited at p. 4, 7, 17, 18, 24]

[158] Ricardo Miranda, F Braunschweig, P Leitao, R Neves, F Martins, and A Santos. Mohid 2000, a coastal integrated object oriented model. *Southampton, UK: WIT Press, Hydraulic Engineering Software VIII*, 2000. [cited at p. 3, 17, 34]

[159] Jeff Moad. Maintaining the competitive edge. *Datamation*, 36(4):61, 1990. [cited at p. 37]

[160] Audris Mockus, Stephen G Eick, Todd L Graves, and Alan F Karr. On measurement and analysis of software changes. *IEEE Transactions on software engineering*, 20, 1999. [cited at p. 29]

[161] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965. [cited at p. 32]

[162] GJ Moridis. TOUGH90: A FORTRAN90 Implementation of TOUGH2. In *TOUGH98 Workshop, Berkeley, CA, May*, 1998. [cited at p. 3, 34]

[163] Chris Morris and Judith Segal. Some challenges facing scientific software developers: The case of molecular biology. In *e-Science, 2009. e-Science'09. Fifth IEEE International Conference on*, pages 216–222. IEEE, 2009. [cited at p. 30]

[164] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. Papi: A portable interface to hardware performance counters. In *Proc. Department of Defense HPCMP Users Group Conference*, 1999. [cited at p. 127]

[165] Glenford J. Myers. *Software reliability - principles and practices.* Wiley, 1976. [cited at p. 72]

[166] Charles D Norton, Viktor K Decyk, and Boleslaw K Szymanski. On parallel object oriented programming in fortran 90. *ACM SIGAPP Applied Computing Review*, 4(1):27–31, 1996. [cited at p. 3, 34]

[167] William L Oberkampf, Timothy G Trucano, and Charles Hirsch. Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews*, 57(5):345–384, 2004. [cited at p. 124]

[168] J. Overbey and C. Rasmussen. Instant IDEs: supporting new languages in the CDT. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, page 79. ACM, 2005. [cited at p. 41]

[169] J. L. Overbey, S. Xanthos, R. Johnson, and B. Foote. Refactorings for Fortran and High-Performance Computing. In *SE-HPCS '05: Proceedings of the second international workshop on Software engineering for high performance computing system applications*, pages 37–39, New York, NY, USA, 2005. ACM. [cited at p. 41]

[170] Jeffrey L Overbey. *A toolkit for constructing refactoring engines*. PhD thesis, Google, Inc, 2011. [cited at p. 78]

[171] Jeffrey L Overbey and Ralph E Johnson. Generating rewritable abstract syntax trees. In *Software Language Engineering*, pages 114–133. Springer, 2009. [cited at p. 75, 117]

[172] Jeffrey L. Overbey and Ralph E. Johnson. Differential precondition checking: A lightweight, reusable analysis for refactoring tools. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011*, pages 303–312. IEEE, 2011. [cited at p. 112]

[173] J.L. Overbey, S. Negara, and R.E. Johnson. Refactoring and the evolution of Fortran. *Urbana*, 51:61801. [cited at p. 41]

[174] J.L. Overbey, S. Negara, and R.E. Johnson. Refactoring and the Evolution of Fortran. In *2nd International Workshop on Software Engineering for Computational Science and Engineering (SECSE'09)*, 2009. [cited at p. 41, 103]

[175] Somsak Phattarsukol and Pornsiri Muenchaisri. Identifying candidate objects using hierarchical clustering analysis. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific*, pages 381–389. IEEE, 2001. [cited at p. 29]

[176] J Pipitone and S Easterbrook. Assessing climate model software quality: a defect density analysis of three models. *Geoscientific Model Development Discussions*, 5(1):347–382, 2012. [cited at p. 31]

[177] Jon Pipitone. *Software quality in climate modelling*. PhD thesis, University of Toronto, 2010. [cited at p. 31]

[178] Constantine D Polychronopoulos. *Automatic restructuring of Fortran programs for parallel execution*. Springer, 1988. [cited at p. 41]

[179] Otis Port et al. The software trap–automate or else. *Business week*, 3051(9):142–154, 1988. [cited at p. 37]

[180] Jane Radatz, Anne Geraci, and Freny Katki. Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990:121990, 1990. [cited at p. 123]

[181] Václav Rajlich and Srikant Varadarajan. Using the web for software annotations. *International Journal of Software Engineering and Knowledge Engineering*, 9(01):55–72, 1999. [cited at p. 29]

[182] Jane Ransom, I Somerville, and Ian Warren. A method for assessing legacy systems for evolution. In *Software Maintenance and Reengineering, 1998. Proceedings of the Second Euromicro Conference on*, pages 128–134. IEEE, 1998. [cited at p. 5]

[183] J. Reid. The new features of Fortran 2003. In *ACM SIGPLAN Fortran Forum*, volume 26, page 33. ACM, 2007. [cited at p. 24]

[184] R Riggs. Computer systems maintenance. *Datamation*, 15(11):227, 1969. [cited at p. 37]

[185] Paul Robertson. Integrating legacy systems with modern corporate applications. *Communications of the ACM*, 40(5):39–46, 1997. [cited at p. 5, 28]

[186] Crispin Rope. Eniac as a stored-program computer: A new look at the old records. *IEEE Annals of the History of Computing*, 29(4):82–87, 2007. [cited at p. 13]

[187] Jean E Sammet and Jerome Garfunkel. Summary of changes in cobol, 1960-1985. *Annals of the History of Computing*, 7(4):342–347, 1985. [cited at p. 17]

[188] Pam Sampson. *On the development of a constructive model for use in software reengineering projects of legacy systems*. ProQuest, 2007. [cited at p. 29]

[189] Miriam Schmidberger and Bernd Brugge. Need of software engineering methods for high performance computing applications. In *Parallel and Distributed Computing (ISPDC), 2012 11th International Symposium on*, pages 40–46. IEEE, 2012. [cited at p. 3, 17, 34]

[190] Mike Schmit. Optimizing pentium code. *Dr Dobb's Journal-Software Tools for the Professional Programmer*, 19(1):40–49, 1994. [cited at p. 125]

[191] Robert C. seacord, Daniel Plakosh, and Grace A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. [cited at p. 37]

[192] Judith Segal. Models of scientific software development. 2008. [cited at p. 31]

[193] Judith Segal. Some challenges facing software engineers developing software for scientists. In *2nd International Software Engineering for Computational Scientists and Engineers Workshop (SECSE '09), ICSE 2009 Workshop, Vancouver, Canada*, pages 9–14, May 2009. [cited at p. 31]

[194] Judith Segal. Scientists and software engineers: A tale of two cultures. *Proceedings of the Psychology of Programming Interest Group, PPIG 08*, pages 10–12, September 2008. [cited at p. 31]

[195] M Shaw et al. Computer science: Reflections on the field, reflections from the field, 2004. [cited at p. 30]

[196] Harry M Sneed. Integrating legacy software into a service oriented architecture. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 11–pp. IEEE, 2006. [cited at p. 30]

[197] E Burton Swanson. The dimensions of maintenance. In *Proceedings of the 2nd international conference on Software engineering*, pages 492–497. IEEE Computer Society Press, 1976. [cited at p. 35]

[198] CMMI Product Team. Capability maturity model® integration (cmmi sm), version 1.1. *Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA, Tech. Rep. SEI-2002-TR-012*, 2002. [cited at p. 123]

[199] J.M. Tendler, J.S. Dodson, JS Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, 2002. [cited at p. 33]

[200] Fernando G Tinetti, Sergio M Martin, Fernando E Frati, and Mariano Méndez. Optimization and parallelization experiences using hardware performance counters. [cited at p. 312]

[201] Fernando G Tinetti and Mariano Méndez. Fortran legacy software: source code update and possible parallelisation issues. In *ACM SIGPLAN Fortran Forum*, volume 31, pages 5–22. ACM, 2012. [cited at p. 41, 257, 261, 312, 313]

[202] Fernando G Tinetti and Mariano Méndez. An automated approach to hardware performance monitoring counters. In *Computational Science and Computational Intelligence (CSCI), 2014 International Conference on*, volume 1, pages 71–76. IEEE, 2014. [cited at p. 312]

[203] Fernando G Tinetti, Mariano Méndez, and Armando De Giusti. Restructuring fortran legacy applications for parallel computing in multiprocessors. *The Journal of Supercomputing*, 64(2):638–659, 2013. [cited at p. 41, 257, 261, 311, 313]

[204] Rémi Triolet, Paul Feautrier, and François Irigoin. Automatic parallelization of fortran programs in the presence of procedure calls. In *ESOP 86*, pages 210–222. Springer, 1986. [cited at p. 41]

[205] John W. Tukey. The teaching of concrete mathematics. 65(1):1–9, January 1958. This article is believed to contain the first published instance of the word 'software' in the meaning of instructions to a computer: "Today the 'software' comprising the carefully planned interpretive routines, compilers, and other aspects of automative programming are at least as important to the modern electronic calculator as its 'hardware' of tubes, transistors, wires, tapes and the like." [p. 2]. [cited at p. 3]

[206] Gernot Urschler. Automatic structuring of programs. *IBM Journal of Research and Development*, 19(2):181–194, 1975. [cited at p. 39]

[207] Giuseppe Visaggio. Ageing of a data-intensive legacy system: symptoms and remedies. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(5):281–308, 2001. [cited at p. 5, 28]

[208] John Von Neumann. The point source solution. *Bethe [Bet47]*, 1941. [cited at p. 13]

[209] John F Wakerly. *Digital design principles and practices*. Prentice-Hall, Inc., 1989. [cited at p. 125]

[210] MP Ware, F George Wilkie, and Mary Shapcott. The application of product measures in directing software maintenance activity. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):133–154, 2007. [cited at p. 37]

[211] Ian Warren. *The Renaissance of Legacy Systems*. Springer, 1999. [cited at p. 5, 29]

[212] L. Washington. Cobol: The new latin. this 'dead language' must be embraced and taught. *ComputerWorld*, November 13,2006. [cited at p. 38]

[213] Arthur H Watson, Thomas J McCabe, and Dolores R Wallace. Structured testing: A testing methodology using the cyclomatic complexity metric. *NIST special Publication*, 500(235):1–114, 1996. [cited at p. 86, 87]

[214] Hugh J Watson. An empirical investigation of the use of simulation. *Simulation & Games*, 1978. [cited at p. 3, 17, 34]

[215] Spencer Weart. The development of general circulation models of climate. *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, 41(3):208–217, 2010. [cited at p. 14, 15, 31]

[216] Gregory V Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5, 2006. [cited at p. 31]

[217] Norman M Wolcott. *FORTRAN IV enhanced character graphics*. Number 32. Dept. of Commerce, National Bureau of Standards, Institute for Computer Sciences and Technology: for sale by the Supt. of Docs., US Govt. Print. Off., 1978. [cited at p. 157]

[218] Takanobu Yamashina, Hidetake Uwano, Kyohei Fushida, Yasutaka Kamei, Masataka Nagura, Shinji Kawaguchi, and Hajimu Iida. Shinobi: A real-time code clone detection tool for software maintenance. *Nara Institute of Science and Technology*, 2008. [cited at p. 26]

[219] Andy Zaidman, Bram Adams, and Kris De Schutter. Applying dynamic analysis in a legacy context: An industrial experience report. *Comprehension through Dynamic Analysis*, page 6, 2005. [cited at p. 26]

[220] M.V. Zelkowitz, A.C. Shaw, and J.D. Gannon. *Principles of software engineering and design*. Prentice Hall Professional Technical Reference, 1979. [cited at p. 5, 7, 37]

[221] M.V. Zelkowitz, A.C. Shaw, and J.D. Gannon. *Principles of software engineering and design*. Prentice Hall Professional Technical Reference, 1979. [cited at p. 89]

[222] Bo Zhang, Liang Bao, Rumin Zhou, Shengming Hu, and Ping Chen. A black-box strategy to migrate gui-based legacy systems to web services. In *Service-Oriented System Engineering, 2008. SOSE'08. IEEE International Symposium on*, pages 25–31. IEEE, 2008. [cited at p. 30]

[223] Zhuopeng Zhang, Ruimin Liu, and Hongji Yang. Service identification and packaging in service oriented reengineering. In *SEKE*, volume 5, pages 620–625, 2005. [cited at p. 30]

Índice de figuras