Universidad de las Ciencias Informáticas

Facultad 10

MED: Módulo para la edición y depuración de código Octave integrado al ambiente EIDMAT

TRABAJO DE DIPLOMA EN OPCIÓN AL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores: Yurenia Hernández Blanco

Eduardo Alejandro Cuesta Llanes

Tutor: MSc. Alexeis Companioni Guerra

Ciudad de la Habana

Mayo 2010

"Hasta donde la ley de las matemáticas se refiere a la realidad, esta no es exacta; y cuando las leyes de la matemática son exactas, estas no se refieren a la realidad."

Albert Einstein

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firman la presente a los _____ días del mes de _____ de 2010.

Yurenia Hernández Blanco

Eduardo Alejandro Cuesta Llanes

MSc. Alexeis Companioni Guerra

Datos de contacto

Alexeis Companioni Guerra, graduado de Ingeniero Nuclear en el año 2002, se incorporó a trabajar en la Universidad de las Ciencias Informáticas (UCI). Posee categoría docente de profesor Asistente y grado científico de Máster en Ciencias. Desde su incorporación a la UCI mantiene una labor científica activa como miembro del Grupo de Matemática y Física Computacionales (GMFC) y conserva vínculos estrechos con grupos de investigación de otras instituciones docentes. En los últimos cuatro años ha trabajado fundamentalmente las temáticas de los sistemas dinámicos, el diagnóstico de fallas en sistemas tanto lineales como no lineales y el tratamiento del ruido en series temporales no lineales obteniendo como resultado varias publicaciones de carácter nacional e internacional así como varias presentaciones en eventos, seminarios científicos y una patente de software. En el año 2003 obtuvo el Premio Nacional a Jóvenes Investigadores que otorga el CITMA a jóvenes de destacada trayectoria investigativa.

Agradecimientos

A nuestros queridos padres, a quienes debemos la vida,
los que con su esmero hicieron posible este sueño.

A nuestro tutor, por el tiempo invertido y la guía aportuna.

A todos los que de una u otra forma han colaborado en
la realización de este trabajo.

Dedicatoria

A nuestros seres queridos.

A la Revolución.

A Fidel.

Resumen

En la actualidad se está llevando a cabo el proceso de migración de los sistemas informáticos cubanos a Software Libre, con el primordial objetivo de alcanzar la necesaria independencia tecnológica en la isla. En aras de facilitar esta trascendental transformación en el país se están desarrollando herramientas de licencias libres que sustituyan a sus homólogas privativas.

En la presente investigación se expone una aplicación para la edición y depuración de código Octave integrada al Entorno Integrado para el Desempeño Matemático (EIDMAT), una interfaz gráfica para el asistente matemático Octave, la alternativa libre de Matlab. Con la que se pretende dotar al EIDMAT de un módulo que agilice todo el proceso de edición y depuración de archivos de código Octave.

Palabras claves

EIDMAT, Octave, Software Libre.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

Las Matemáticas son tan antiguas como la propia humanidad. Ciencia que remonta sus orígenes a más de 20 000 años y que surge por la necesidad práctica de contar objetos en las grandes civilizaciones, que en sus inicios sólo contaban con los medios disponibles como las piedras y los dedos.

Grandes civilizaciones antiguas alcanzaron un desarrollo considerable en la práctica de esta disciplina, primitiva en sus inicios y que fue evolucionando según las necesidades imperantes[1]. Las primeras referencias datan en Egipto al desarrollar el "sistema de numeración jeroglífico". La civilización mesopotámica, por su parte, dio su gran tributo en el campo de la potenciación. Aunque fueron notorias las contribuciones de estas culturas, ninguna aportó tanto como las surgidas a lo largo del Mediterráneo. Pues de Tales de Mileto a Euclides de Alejandría cimentaron una superpotencia invencible y única, que perdura hasta nuestros días y es conocida como Matemáticas.

Por la gran utilidad en la resolución de problemas de toda naturaleza, el hombre extendió el uso de esta rama a todas las esferas de la vida; no obstante, la complejidad de los problemas que se presentaban en la práctica continuó en ascenso y los mecanismos existentes se veían limitados para resolverlos imponiéndose así un desarrollo continuo en las formas de hacer, de ver la vida, y aparejado a esto de las herramientas de cálculo. Impulsado por esta carrera científica aliñada con otros descubrimientos de importancia capital, tiene lugar el nacimiento de una de las invenciones más pródigas del hombre: el ordenador. Aunque en sus inicios fueron las calculadoras de relojería de Pascal y Leibniz, ya Charles Babbage en el siglo XIX

diseñó una máquina capaz de realizar operaciones matemáticas automáticamente siguiendo una lista de instrucciones.

Con el paso del tiempo los ordenadores han evolucionado hasta convertirse en pequeños procesadores de bolsillo capaces de resolver algoritmos extremadamente complejos. Unido a esto se han desarrollado programas computacionales que facilitan la solución y el aprendizaje de complicados problemas matemáticos, como son los asistentes matemáticos. Estos tienen un gran potencial en la realización de cálculos, tanto numéricos como simbólicos. Su uso contribuye a que el individuo progrese hacia niveles superiores de pensamiento formal, favorecen la interiorización de conceptos y procedimientos matemáticos.

Una de las herramientas fundamentales en los asistentes matemáticos lo constituyen los editores depuradores, ya que facilitan y agilizan la edición y depuración de código. Su surgimiento es producto de la evolución de los editores de líneas, en los que era posible la edición de una sola línea a la vez. Y luego con la aparición del monitor de tubos de rayos catódicos como dispositivo de visualización fue posible la edición a "pantalla completa". Cuando los editores de pantalla completa incluyen compiladores e intérpretes, se les nombra *editores depuradores*. Esto significa que el código puede ser escrito, editado, compilado y ejecutado desde el mismo programa.

Uno de los asistentes matemáticos de esencia numérica más utilizado en la actualidad es el Matlab, este cuenta con un poderoso editor depurador que permite ver, desarrollar y depurar de forma rápida los programas realizados empleando este lenguaje. A pesar de todas las prestaciones que ofrece esta potente herramienta de cálculo, no es un secreto que es una aplicación privativa y su uso conlleva a una dependencia con consecuencias casi irreversibles. Por otra parte, y no menos importante, es de señalar que esta aplicación no se encuentra al alcance de la gran mayoría de los usuarios dado los elevados precios que posee.

En el caso particular de Cuba el embargo económico y político impuesto por parte del Go-

bierno de los Estados Unidos, "permite" piratear las aplicaciones; puesto que las leyes de la propiedad intelectual vigentes en la constitución de los Estados Unidos no pueden ser aplicadas a este país. Pero si es muy probable que cualquier empresa de software propietario pueda lanzar una campaña de difamación moral por usar este tipo de software, lo que sería una derrota para la nación que preserva los principios más altruistas que existen[2].

Por estas razones, es perfectamente entendible la necesidad imperiosa de contribuir a aplicaciones de software libre, con el objetivo de dotarlas de todas las prestaciones hoy disponibles en las privativas. Situaciones como esta han sido las impulsoras para la creación de potentes herramientas libres para el cálculo. Un ejemplo fehaciente es el asistente matemático Octave, cuyo nacimiento nada tiene que ver con Matlab, pero por cuestiones de necesidad ha ido convergiendo hacia la compatibilidad y convirtiéndose en la alternativa libre al mismo, colocándose al alcance de una mayor cantidad de usuarios.

Octave es un software destinado a la resolución de problemas relacionados con el cálculo numérico; no obstante, posee la desventaja de ser una aplicación orientada a línea de comandos por lo que no es muy presta al aprendizaje sencillo o intuitivo como lo puede ser una aplicación con interfaz gráfica. Siguiendo esta línea de necesidad se han desarrollado proyectos como Koctave, Joctave, Goctave, YAOG, Octivate, Octave Workshop y QtOctave[1], con el objetivo de dotar a esta potente herramienta de un entorno de desarrollo integrado, que brinde una mayor interactividad a los usuarios. Aunque algunas de estas creaciones han aportado bastante al uso de Octave, todas han tenido la deficiencia de no proveer al usuario una herramienta que facilite todo el proceso de edición y depuración de código; lo que es fundamental para que los usuarios puedan ver, desarrollar y depurar de forma rápida los programas en este lenguaje.

Otro de los intentos por dotar a Octave de una interfaz gráfica, surgido al calor de la búsqueda de soluciones libres y que intenta limar todas las deficiencias encontradas en los esfuerzos anteriores, es el Entorno Integrado para el Desempeño Matemático (EIDMAT). Actualmente

EIDMAT consta de una interfaz principal, que provee una ventana de comandos, mediante la que el usuario interactúa con Octave de una forma simple e intuitiva (Ver Anexo 1). Cuenta con un historial que mantiene un registro de los comandos usados para su posterior reutilización (Ver Anexo 2), entre otras posibilidades (Ver anexos 3, 4 y 5). A pesar de todas las prestaciones que ofrece hoy EIDMAT, el proceso de edición y depuración de código Octave se hace complejo para los usuarios del mismo dado que se necesita conocer cada uno de los comandos del lenguaje. De manera adicional, para lograr este proceso con éxito es necesario realizar una gran cantidad de acciones que provocan que el mismo sea muy lento y tedioso por lo que constituye una limitante para el desempeño de los usuarios sobre dicha plataforma.

Problema de investigación

¿Cómo agilizar el proceso de edición y depuración de código Octave de modo que posibilite a los usuarios un mejor desempeño sobre la plataforma EIDMAT?

Objeto de estudio

Editores depuradores.

Campo de acción

Editores depuradores de código Octave.

Objetivo general

Desarrollar un módulo que agilice la edición y depuración de código Octave sobre la plataforma EIDMAT.

Objetivos específicos

- Analizar los editores de código existentes que pueden ser utilizados para la edición y depuración de código Octave.
- Construir un editor depurador de código Octave integrado a la plataforma EIDMAT.
- Validar los resultados obtenidos.

Tareas de investigación

- Revisión y análisis de la bibliografía existente relacionada con los editores de código Octave.
- Selección de la metodología de desarrollo y las tecnologías adecuadas para el desarrollo de la aplicación.
- Realización del diseño del sistema a implementar.
- Implementación del sistema propuesto.
- Ejecución de los casos de prueba de aceptación para asegurar la calidad del producto.

Idea a defender

El desarrollo de un módulo para la edición y depuración de código Octave agilizará dicho proceso sobre la plataforma EIDMAT.

Métodos científicos

El **Histórico-Lógico:** con el propósito de analizar la trayectoria completa de los editores depuradores y su condicionamiento a los diferentes períodos de la historia. El **Analítico-Sintético:** con el objetivo de buscar la esencia de los editores depuradores, los rasgos que los caracterizan y los distinguen, mediante el análisis de las teorías y documentos existentes,

permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio.

Resultados esperados

Obtención de un informe con toda la base teórica sobre la cual está sustentada la solución propuesta. Así como el módulo integrado a la plataforma EIDMAT que agilizará el proceso de edición y depuración de código Octave.

Estructura del documento

El presente informe está compuesto por la introducción, tres capítulos, conclusiones, referencias bibliográficas, bibliografía, anexos y glosario de términos, donde se expone y da cumplimiento de forma progresiva y convincente a la totalidad de los objetivos propuestos:

Capítulo 1: Fundamentación teórica.

En este capítulo se definen algunos conceptos, como es el caso de editor de texto, editor de código y depurador, esenciales para la comprensión del tema tratado. Se hace un análisis de las principales tecnologías actuales relacionadas con los editores depuradores de código Octave. Además se documentan la metodología de desarrollo, el lenguaje y las herramientas a utilizar en el desarrollo del sistema propuesto.

Capítulo 2: Planificación y desarrollo de la propuesta de solución.

En este capítulo se inicia el desarrollo de la propuesta de solución haciendo uso de la metodología seleccionada. Se presentan los principales artefactos generados en las primeras fases que esta propone.

Capítulo 3: Validación de la solución propuesta.

En este capítulo se presentan los casos de pruebas de aceptación a los que fue sometida la aplicación durante su ciclo de desarrollo. Se muestran los resultados obtenidos y se exponen las funcionalidades alcanzadas durante el período de desarrollo.

Capítulo 1

Fundamentación teórica

El objetivo fundamental de este capítulo consiste en esclarecer los aspectos más importantes que sirven de soporte teórico a la construcción del sistema. Se brinda además una visión general de los conceptos vinculados con los editores depuradores, imprescindibles para comprender el tema tratado a la vez que se expone, como elemento esencial, un estudio del estado del arte sobre los editores de código Octave hoy publicados en el mundo. Por úlimo se presentan tanto la metodología como herramientas y tecnologías que se a utilizar en la implementación del sistema.

1.1. Conceptos fundamentales

1.1.1. Software libre

La Fundación de Software Libre es una organización creada en Octubre de 1985 a partir del esfuerzo de Richard Matthew Stallman y otros entusiastas del software libre con el propósito de difundir este movimiento. El "Software Libre" es un asunto de libertad, no de precio. Para entender el concepto, se debe pensar en "libre" como en "libertad de expresión", lo que ha dado lugar a cierta confusión[3].

Se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar

y mejorar el software, o sea, especialmente a cuatro clases de libertad para los usuarios de software[4]:

Libertad 0: la libertad para ejecutar el programa, con cualquier propósito.

Libertad 1: la libertad para estudiar el funcionamiento del programa y adaptarlo a tus necesidades.

Libertad 2: la libertad para redistribuir copias y ayudar así a tu vecino.

Libertad 3: la libertad para mejorar el programa y luego publicarlo para el bien de toda la comunidad.

1.1.2. Software privativo

Cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo, o que su código fuente no está disponible o el acceso a este se encuentra restringido.

1.1.3. Licencia Pública General (GPL)

GNU[1] GPL es una licencia creada por la Fundación de Software Libre a mediados de los años ochenta, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

1.1.4. Octave

GNU Octave es un lenguaje de programación de alto nivel. Está diseñado para la solución de problemas numéricos. Software completamente gratuito basado en la filosofía de GNU, por lo que se tiene acceso al programa y al código fuente del mismo para transformarlo si se desea.

[1] Acrónimo recursivo que significa GNU No es Unix.

Octave posee una gran cantidad de herramientas que permiten resolver problemas de algebra lineal, cálculo de raíces de ecuaciones no lineales, integración de funciones ordinarias, manipulación de polinomios, integración de ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas. Sus funciones también se pueden extender mediante funciones definidas por el usuario escritas en el lenguaje propio de Octave o usando módulos dinámicamente cargados escritos en lenguajes como C, C++ y Fortran entre otros[5].

1.1.5. Entorno de Desarrollo Integrado (IDE)

Aplicación gráfica que incluye intérprete, consola interactiva, editor, cronología de comandos, visor de documentación, gestor de archivos y depurador. Se hace con la intención de que no se tenga que utilizar ninguna aplicación externa. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

1.1.6. Depuración de programas

Es el proceso de identificar y corregir errores de programación. En inglés se le conoce como debugging, ya que se asemeja a la eliminación de bichos (bugs), manera en que se conoce informalmente a los errores de programación. Se dice que el término bug proviene de la época de las computadoras de válvula termoiónicabulbos, en las cuales los problemas se generaban por los insectos que eran atraídos por las luces y estropeaban el equipo.

1.1.7. Punto de ruptura

Es una línea de código en la que el depurador interrumpe la ejecución de un programa informático. Cuando se detiene la ejecución, el programador es capaz de revisar el estado y valores de todas las variables que están utilizándose en esos momentos en el programa. Permite entender qué es lo que pueda estar pasando y qué pueda dar lugar a errores.

El punto de ruptura se establece habitualmente con un entorno de desarrollo integrado (IDE), que contiene un depurador o hace uso de él. El programador es el encargado de decidir donde

establecer los puntos de ruptura, señalando la línea o líneas en el código donde quiere detener la ejecución.

1.2. Editor de texto

Un editor de texto es un programa que permite crear y modificar archivos digitales compuestos únicamente por texto sin formato, conocidos comúnmente como archivos de texto plano. Son incluidos en el sistema operativo o en algún paquete de software instalado y se usan cuando se deben crear o modificar archivos de texto, como archivos de configuración o el código fuente de algún programa.

Algunos editores son sencillos, mientras que otros ofrecen una amplia gama de funciones como pueden ser:

- Editores de código fuente, diseñados para un lenguaje de programación determinado, con coloreado de sintaxis, macros, completamiento de palabras.
- Editores con regiones plegables, a veces no todo el texto es relevante para el usuario. Con este tipo de editores ciertas regiones con texto irrelevante pueden ser plegadas o escondidas, mostrando al usuario sólo lo importante del texto.
- Entorno de Desarrollo Integrado, es un editor más otras herramientas de trabajo, compiladores, extractores de diferencias entre dos textos, repositorios, incluidos en un mismo programa.

1.2.1. Funciones típicas de un editor de texto

Marcar región: Es la función que marca, visualmente o no, una parte del texto para ser elaborada con otras funciones. La región puede contener varias líneas del texto o bien varias columnas adyacentes del texto.

Búsqueda y reemplazo: El proceso de búsqueda de una palabra o una cadena de caracteres, en un texto plano y su reemplazo por otra. Existen diferentes métodos: global, por región, reemplazo automático, reemplazo con confirmación, búsqueda de texto o búsqueda de una expresión regular.

Copiar, cortar y pegar: Útil para copiar, trasladar o borrar una región marcada.

Formatear: Los editores de texto permiten automatizar las únicas funciones de formateo que utilizan: quebrar la línea, indentar, formatear comentarios o formatear listas.

Deshacer y rehacer: Consiste en que el programa editor va almacenando cada una de las operaciones hechas por el usuario hasta un número configurable. Si el usuario se arrepiente de algún cambio, por muy anterior que sea, el editor le permite revertir todos los cambios hechos hasta el número configurado. Rehacer es por consiguiente, revertir algo revertido.

Importar: Agregar o insertar el contenido de un archivo, en el archivo que se está editando.

Filtros: Los editores de texto permiten hacer pasar las líneas del texto o de una región por algún programa para modificarlas u ordenarlas. Por ejemplo, para ordenar alfabéticamente una lista de nombres.

1.3. Editores de código fuente

Es un editor de texto diseñado específicamente para editar el código fuente de programas informáticos. Estos poseen características concebidas exclusivamente para simplificar y acelerar la escritura de código tales como resaltado de sintaxis, completamiento y pareo de llaves. Además proveen un modo conveniente de ejecutar un depurador, o cualquier otro programa que sea relevante en el proceso de desarrollo de software por lo que si bien muchos editores de texto pueden ser usados para editar código fuente sin dificultades, entre tanto no automatizan y facilitan la edición del código, no son considerados “editores de código fuente”.

Esta herramienta de suma importancia para el programador tiene funciones específicamente dedicadas a la programación como son:

- Verifican la sintaxis a medida que el programador escribe, alertando inmediatamente sobre los problemas de sintaxis que puedan surgir.
- Comprimen el código, convirtiendo las palabras claves en tokens[2] de un solo byte, eliminando espacios en blanco innecesarios y convirtiendo los números a una forma binaria. Estos editores descomprimen el código fuente al momento de visualizarlo, imprimiéndolo con los espacios y mayúsculas adecuadas.

Ejemplos de editores de código fuente:

- ▷ Eclipse.
- ▷ Emacs (multiplataforma, incluyendo Unix, Linux, Mac OS X, Windows).
- ▷ SciTE.
- ▷ Vi/Vim (multiplataforma, incluyendo Unix, Linux, Mac OS X, Windows).
- ▷ Kate (KDE[3]).

1.4. Depurador

Es un programa pensado para la localización y reparación de errores en el código fuente de una aplicación. Un depurador permite ver el contenido de las variables, a medida que se ejecuta el código, y establecer puntos de ruptura o detención de la ejecución para ver los datos con los que se está trabajando. Son particularmente útiles cuando el programa parece estar bien, pero no da el resultado esperado. El depurador se basa en los puntos de ruptura, donde se detiene la ejecución del programa, permitiendo al usuario[6]:

- Examinar y modificar la memoria y las variables del programa.

[2]Son el equivalente a las palabras y signos de puntuación en el lenguaje natural escrito.
[3]Del inglés K Desktop Environment.

- Examinar el contenido de los registros del procesador.
- Examinar la pila de llamadas que han desembocado en la situación actual.
- Cambiar el punto de ejecución, de manera que el programa continúe su ejecución en un punto diferente al punto en el que fue detenido.
- Ejecutar instrucción a instrucción.
- Ejecutar partes determinadas del código, como el interior de una función, o el resto de código antes de salir de una función.

El depurador permite detener el programa en:

- Un punto determinado mediante un punto de ruptura.
- Un punto determinado bajo ciertas condiciones mediante un punto de ruptura condicional.
- Un momento determinado cuando se cumplan ciertas condiciones.
- Un momento determinado a petición del usuario.

1.5. Editores de código para Octave de uso actual

En Cuba son pocas las soluciones que se podrían encontrar hasta el momento, debido al escaso desarrollo de aplicaciones de este tipo dentro de la naciente Industria del Software. Después de una búsqueda exhaustiva, el estudio se ha basado en un grupo de editores de código existentes en el mundo, que pueden ser utilizados para la edición y depuración de código Octave. Los candidatos a un análisis riguroso se muestran a continuación.

1.5.1. Bluefish

Bluefish es un potente editor para programadores. Soporta múltiples lenguajes pero se centra en la edición de páginas dinámicas e interactivas (Ver Figura 1.1). Emplea principalmente las bibliotecas GTK y C posix. Cuenta con características tales como:

- Rapidez.
- Posibilidad de abrir múltiples archivos simultáneamente.
- Soporte multiproyecto.
- Numeración de líneas.
- Marcado de sintaxis personalizable basado en expresiones regulares.

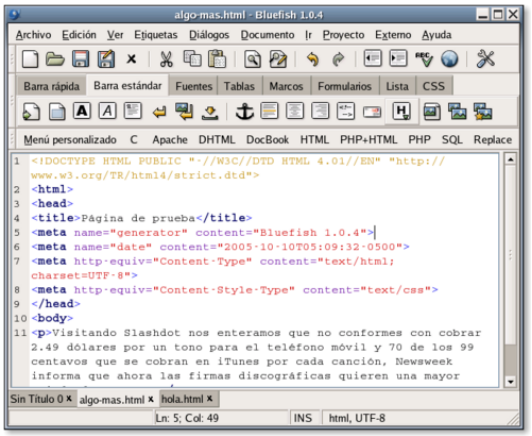


Figura 1.1: Bluefish.

Desventajas:

- No permite la depuración del código Octave.
- La configuración del programa es poco intuitiva, sobre todo el sistema de coloreado de sintaxis.
- Carencia de plegado de código.
- No posee resaltado de la línea actual.
- Ausencia de completado de código.
- No reconoce errores de sintaxis.

1.5.2. Gedit

Editor de textos libre que se distribuye bajo licencia GPL. Se caracteriza principalmente por su facilidad de uso, gracias a su interfaz gráfica clara y limpia (Ver Figura 1.2). Muestra únicamente las funcionalidades principales que suelen requerir la mayoría de usuarios como copiar, cortar y pegar texto. Gedit incorpora, entre otras, las funcionalidades siguientes:

- Soporte de textos internacionalizados, usando la codificación UTF-8[4].

[4]Es una norma de transmisión de longitud variable para caracteres codificados utilizando Unicode.

- Resaltado de sintaxis.
- Incorporación de plugins para ampliar las funcionalidades básicas del programa.
- Completamiento de código.
- Posibilidad de cambiar el color y fuente del texto.
- Numeración de líneas.
- Búsqueda y reemplazo de texto.
- Edición de archivos remotamente.
- Copia de seguridad de los ficheros sobre los que se trabaja.
- Edición múltiple en pestañas.

Desventajas:

- No resalta errores sintácticos.
- No posee plegado de código.
- Es un poco pesado para máquinas de pocos recursos.
- No permite la depuración del código Octave.

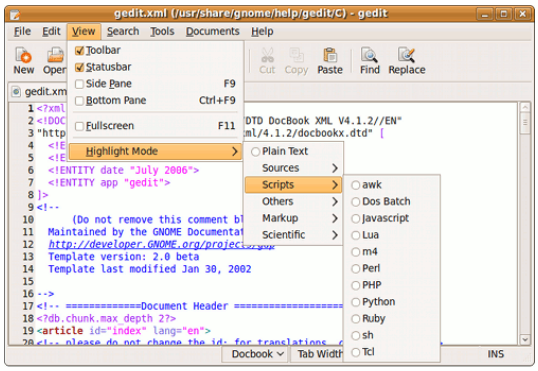


Figura 1.2: Gedit.

1.5.3. SciTE

Editor de texto multiplataforma escrito por Neil Hodgson usando el componente de edición Scintilla. Ligero y hecho para ser veloz, está diseñado principalmente para edición de código fuente, y resaltado de sintaxis (Ver Figura 1.3). Posee múltiples opciones de visualización, búsqueda y reemplazo de palabras, edición y compilación en varios lenguajes de programación.

Aunque es un editor de código bastante completo tiene la limitante de no señalar los errores sintáctico en el código escrito, así como la imposibilidad de depurar y ejecutar el código Octave desde la propia aplicación.

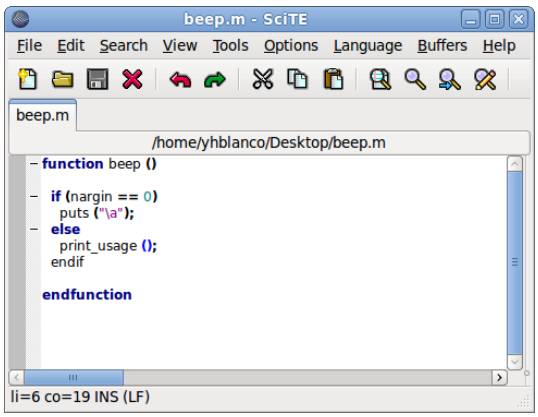


Figura 1.3: SciTE.

1.5.4. Kate

Editor de textos para KDE, ideal para los usuarios que le guste la personalización al máximo (Ver Figura 1.4). El entorno de desarrollo integrado, KDevelop, y la herramienta de desarrollo de páginas web, Quanta Plus, son dos de las más importantes aplicaciones para KDE que usan Kate como componente de edición.

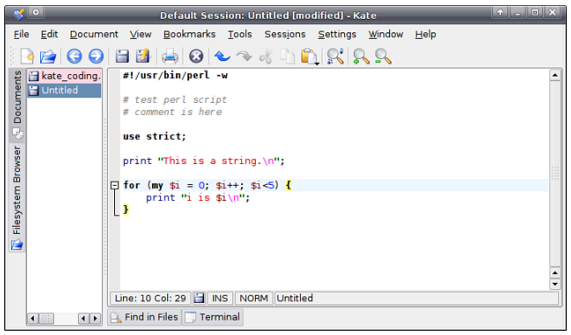


Figura 1.4: Kate.

Entre otras características Kate incluye:

- Resaltado de sintaxis, extensible.
- Búsqueda y remplazo de texto usando expresiones regulares.
- Mantener múltiples documentos abiertos en una ventana.
- Seguimiento de código para C++, C, PHP y otros.
- Soporte de sesiones.
- Manejador de archivos.

- Resaltado y numeración de línea.

Tiene un conjunto de limitantes como son:

- Solo autocompleta si la función ya fue escrita antes en el archivo.
- No permite la depuración de código Octave.
- No resalta errores de sintaxis en el código escrito.

1.5.5. Editor/depurador de QTOctave

Forma parte del Entorno de Desarrollo Integrado (IDE) QtOctave y recopila los principales avances de las interfaces anteriores, creadas para Octave. Dentro de las características que posee se encuentran:

- Permite el resaltado de la sintaxis del lenguaje Octave (Ver Figura 1.5).
- Mantener múltiples documentos abiertos en una ventana.
- Posibilita la ejecución y depuración de código Octave desde la propia aplicación.

Tiene un conjunto de limitantes que dificultan el desempeño de los usuarios sobre la herramienta como son:

- No debería abrir varias veces el mismo archivo o documento, una vez que se encuentre abierto.
- Al cerrar el editor debería preguntar si se guardan los archivos modificados, en caso de que hubiesen.
- No permite la depuración de código mediante el uso de puntos de ruptura.

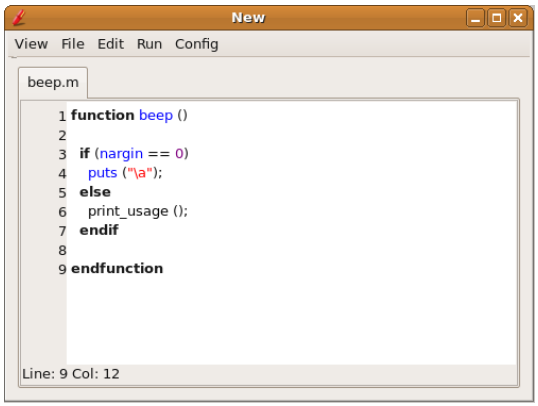


Figura 1.5: QtOctave.

- Tiene elementos de diseño pocos intuitivos al usuario, como es la sustitución de la "x" para cerrar una pestaña en cada una de estas, por un botón en la barra de herramientas, entre otros.

1.5.6. Vim

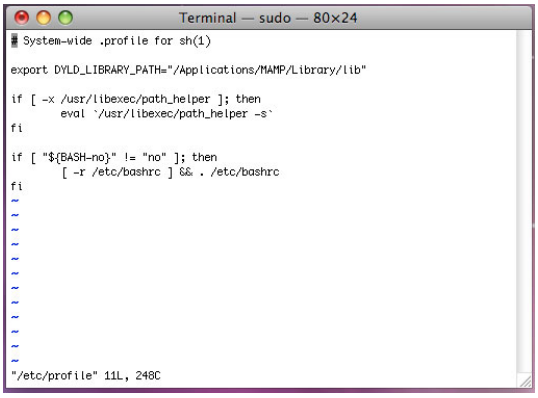


Figura 1.6: Vim.

Su principal característica consiste en que dispone de diferentes modos entre los que se alterna para realizar ciertas operaciones. Esto le diferencia de la mayoría de editores comunes, que tienen un sólo modo en el que se introducen los comandos mediante combinaciones de teclas o interfaces gráficas. Posibilita la edición de código fuente (Ver Figura 1.6).

El trabajo del programador también se ve facilitado por el resaltado de sintaxis y la funcionalidad de plegado de código. Permite la comparación de dos o más ficheros de diferentes versiones, resaltando en cada uno de estos las diferencias existentes entre ambos. Dispone de un lenguaje interpretado para programar nuevas funcionalidades.

Las críticas referentes a este editor radican en que sus modos de operación son completamente diferentes de la mayoría de los editores convencionales actuales. Sobre todo conciernen a la separación de las operaciones en distintos modos, y la necesidad de aprender numerosas combinaciones de teclas, sin las que no es posible trabajar de forma eficiente. Actualmente la falta de empleo del ratón como complemento extra a las combinaciones de teclas se considera como un anacronismo, pues puede llevar al usuario ocasional a la frustración. Sólo tras un aprendizaje prolongado se consigue aumentar la productividad.

1.5.7. GNU Emacs

Es uno de los editores de texto más potentes y versátiles de hoy en día, parte del proyecto GNU. Considerado como un editor extensible, personalizable, auto-documentado y de tiempo real. Utiliza una extensión de lenguaje muy poderosa, Emacs Lisp, que permite manejar tareas distintas, desde escribir y compilar programas hasta navegar en Internet (Ver Figura 1.7). Es mantenido por el Proyecto GNU Emacs, el cual cuenta entre sus miembros a Richard Stallman.

El desarrollo de código Octave se puede facilitar en gran medida usando Emacs con el modo Octave. Un modo muy importante para la edición de archivos Octave (.m), en el que se puede por ejemplo indentar el código automáticamente, resaltar y completar las palabras reservadas del lenguaje. Permite la comunicación con el intérprete Octave, lo que facilita la depuración del código desde la propia aplicación.

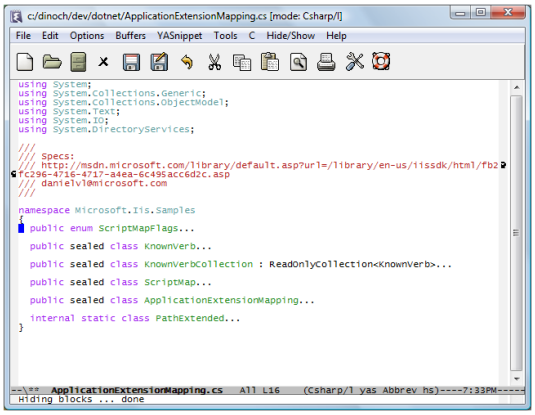


Figura 1.7: Emacs.

A pesar de ser el editor que mejor se complementa con Octave, su uso desde EIDMAT no es factible por las siguientes razones:

- Proceso de depuración complejo, ya que este es llevado a cabo de forma manual, mediante el envío de comandos a Octave.
- Al ser Emacs una aplicación independiente de EIDMAT, el usuario no contaría con todas las funcionalidades presentes en este último.
- La utilización de Emacs conlleva a la ejecución de una nueva instancia de Octave independiente de la que utiliza EIDMAT, lo que trae consigo un aumento considerable de la Memoria de Acceso Aleatorio (RAM).

- Difícil de usar y personalizar para usuarios principiantes.
- No reconoce errores sintácticos en el código escrito.
- No es amigable, su curva de aprendizaje es empinada y no acaba nunca.
- El diseño de Emacs, basado en Lisp, es una penalización de rendimiento, resultante del hecho de cargar e interpretar el código Lisp.

1.6. Metodologías de desarrollo

El desarrollo de software es una tarea difícil de controlar y como resultado a este problema ha surgido una alternativa, las metodologías. Estas imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. El impacto de elegir la mejor metodología para un equipo en un determinado proyecto es trascendental para el éxito del producto.

¿Qué es una metodología de desarrollo?

Una metodología de desarrollo de aplicaciones informáticas es un conjunto de métodos que permiten sistematizar actividades. Estas indican cómo hacer las cosas a través de procedimientos bien descritos[7].

En los últimos tiempos la cantidad y variedad de metodologías de software han aumentado de manera impresionante. Han surgido dos corrientes, los llamados métodos pesados y los métodos ágiles. La diferencia fundamental entre ellos es que, mientras los métodos pesados buscan cumplir el objetivo común mediante el orden y la documentación, los métodos ágiles constituyen una solución a medida para entornos cambiantes, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

1.6.1. Metodologías tradicionales

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas, notaciones para el modelado y documentación detallada.

En el desarrollo de sistemas pequeños o de mediana complejidad, la adopción de una metodología tradicional se percibe como un obstáculo más que como una ventaja, puesto que las ventajas sólo se hacen notorias a largo plazo, y los objetivos primarios de cualquier institución van dirigidos a producir software en el menor tiempo y costo posibles[8].

Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno donde los requisitos no pueden predecirse o bien pueden variar.

1.6.2. Metodologías ágiles

Las metodologías ágiles forman parte del movimiento de desarrollo ágil de software, conocidas anteriormente como metodologías livianas, que se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto. Se le denomina ágil como la habilidad de responder de forma versátil al cambio para maximizar los beneficios. Intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados[9].

Como resultado de esta nueva teoría se crea un Manifiesto Ágil cuyas ideas esenciales son:

- Los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados.

- Es más importante crear un producto software que funcione, que escribir documentación exhaustiva.
- La colaboración con el cliente debe prevalecer sobre la negociación de contratos.
- La capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan.

Debido a que el equipo de desarrollo está compuesto por dos personas, el tiempo disponible es pequeño y se trabaja en un entorno donde los requisitos pueden variar, se decide el uso de una metodología ágil. Con el objetivo de hacer una correcta selección de la que más se adecue a las particularidades del proyecto, se hace un análisis de tres de estas:

- Programación Extrema (XP).
- Scrum.
- SXP (Combinación de XP y Scrum).

1.6.2.1. XP

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios[10].

El desarrollo de software con el uso de XP tiene un conjunto de características que lo distinguen del resto de las metodologías (Ver Figura 1.8 tomada de [11]):

- El cliente o el usuario se convierte en miembro del equipo.
- Se obtiene retroalimentación de usuarios y clientes desde el primer día.

- El software es liberado en entregas frecuentes tan pronto como sea posible.
- Los cambios se implementan rápidamente tal y como fueron sugeridos.
- Las metas en características, tiempos y costos son reajustadas permanentemente en función del avance real del proyecto.
- Comienza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- No introduce funcionalidades antes que sean necesarias.

Ciclo de desarrollo de XP[12]:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

Ventajas:

- Apropiado para entornos inestables.
- La capacidad de respuesta ante un cambio, significa reducir su coste.
- Planificación transparente para los clientes.
- Permite definir en cada iteración cuales son los objetivos de la siguiente.
- Posibilita la realimentación de los usuarios.
- La presión está a lo largo de todo el proyecto y no en una entrega final.

Desventajas:

- Delimitar el alcance del proyecto con el cliente (para mitigar esta desventaja se plantea definir un alcance a alto nivel basado en la experiencia).

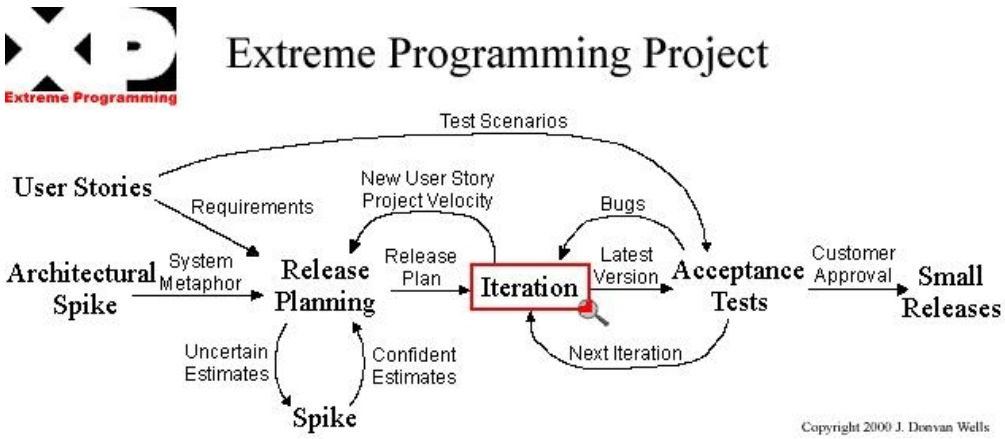


Figura 1.8: Esquema de trabajo XP.

1.6.2.2. Scrum

Scrum es un proceso ágil y liviano que sirve para administrar y controlar la producción de software. El desarrollo se realiza en forma iterativa e incremental. Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. Las iteraciones en general tienen una duración entre 2 y 4 semanas. Scrum se utiliza como marco para otras prácticas de Ingeniería de Software como RUP o XP (Ver Figura 1.9 tomada de [13]).

En Scrum, el equipo se focaliza en una única cosa: construir software de calidad. Por el otro lado, la gestión de un proyecto, Scrum se centra en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en remover cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

En Scrum se promueven valores como:

- Equipos auto-dirigidos y auto-organizados. No hay Director que decida; la excepción es el Scrum Master que debe ser por ciento programador y el que soluciona los problemas.
- Una vez elegida una tarea, no se agrega trabajo extra. En caso que se agregue algo, se recomienda quitar alguna otra cosa.
- Encuentros diarios donde se hacen las preguntas siguientes[14]:
  - ¿Qué es lo que se hizo el día anterior?
  - ¿Qué es lo que se va a hacer hoy?
  - ¿Qué impedimentos tengo para realizar mi trabajo?
- Iteraciones de treinta días, se admite que sean más frecuentes.
- Al principio de cada iteración, planeamiento adaptativo guiado por el cliente.
- Demostración a participantes externos al fin de cada iteración.

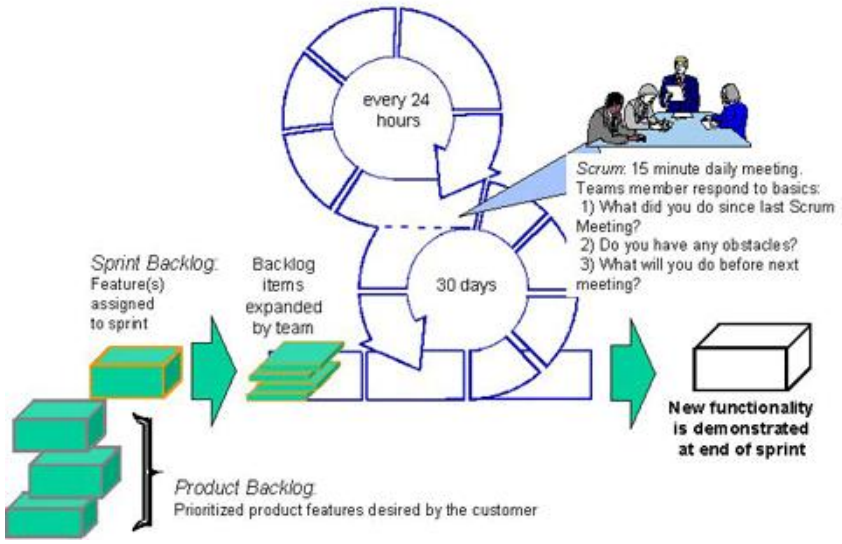


Figura 1.9: Esquema de trabajo Scrum.

1.6.2.3. SXP

SXP está compuesta por las las mejores prácticas de las metodologías XP y Scrum. Propuesta en el 2008 por la ingeniera Gladys Marsi Peñalver Romero; aprobada y puesta en práctica en varios de los proyectos que trabajan con Software Libre, en la Facultad 10 de la Universidad de la Ciencias Informáticas.

Especialmente indicada para proyectos de pequeños equipos de trabajo, rápido cambio de requisitos o requisitos imprecisos. Donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una elevada flexibilidad.

En dicha metodología Scrum juega un papel fundamental en la parte de la planificación del proyecto, debido a que es una forma de gestionar proyectos de software, no es una metodología de análisis, ni de diseño, sino más bien una metodología de gestión del trabajo. Mientras que de XP se aprovecha su concepción de estar encaminada al desarrollo. Consiste en una programación extrema, cuya peculiaridad es tener como parte del equipo al usuario final.

Las cuatro fases definidas en SXP son (Ver Figura 1.10 tomada de [11]):

- **Planificación-Definición:** Se define la concepción inicial del sistema a desarrollar, establece la visión y se realiza el aseguramiento del proyecto.
- **Desarrollo:** Se realiza la implementación del sistema.
- **Entrega:** Puesta en marcha.
- **Mantenimiento:** Soporte para el cliente.

En fin, con la utilización de SCRUM para la gestión se logra una planificación y organización inigualable; mientras que XP respalda con sus prácticas todo el proceso de desarrollo, obteniéndose de esta forma un proceso de software completo[11].

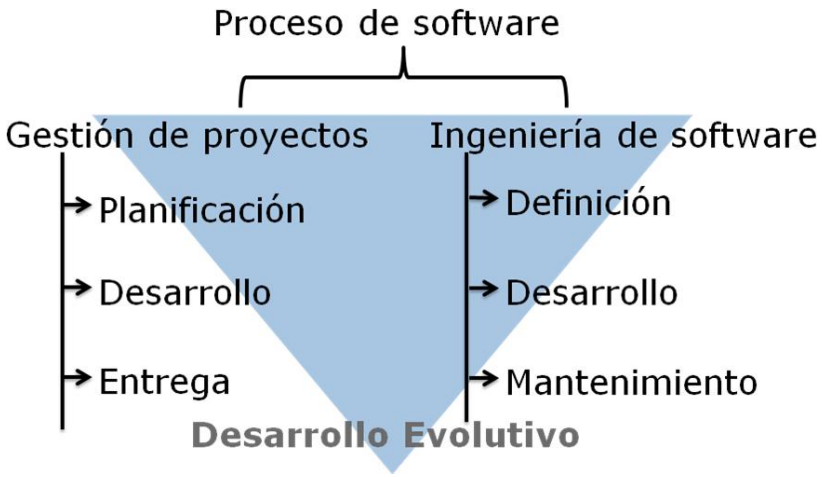


Figura 1.10: Esquema de trabajo SXP

1.6.3. Selección de la metodología a utilizar

A partir del análisis anteriormente expuesto, se determina que SXP guiará el desarrollo del software. Esta hace una recopilación de las mejores prácticas de dos metodologías muy utilizadas en la actualidad y además se adapta a las características propias del proyecto. Ha sido puesta en práctica en el desarrollo de sistemas similares en la facultad 10 de la Universidad de las Ciencias Informáticas, donde se han obtenido resultados satisfactorios, precisamente, porque fue propuesta basada en las peculiaridades de los proyectos de esta área, de la que el sistema a desarrollar forma parte.

1.7. Herramientas, tecnologías y lenguaje a utilizar

Después de entendida y asimilada la metodología de desarrollo a utilizar, lo que ha reducido los riesgos de seleccionar una herramienta inadecuada. Se desarrolla un proceso formal de evaluación crítica de las herramientas, tecnologías y lenguajes propuestos, con el objetivo de reducir la posibilidad de adquirir un producto inadecuado o innecesario. Quedando seleccionadas las que se presentan a continuación.

1.7.1. Python

Lenguaje de programación creado por Guido van Rossum en el año 1991[15]. Es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa[16].

Características:

- Python es un lenguaje de programación multiparadigma. Esto significa que permite varios estilos: programación orientada a objetos, programación estructurada, entre otros.
- Usa tipos de dato dinámico y conteo referencial[5] para el manejo de memoria.
- Tiene una gran biblioteca estándar, usada para una diversidad de tareas.
- En Python, todo es un objeto (incluso las clases).
- Soporta herencia múltiple y polimorfismo.
- El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño.
- Posee muchas cualidades para seguir escalando dentro de los lenguajes de programación más usados en el desarrollo de Software Libre.
- Es un lenguaje potente, seguro, flexible, pero con una gran cantidad de módulos para todas las necesidades que pueden ser útiles a la hora de programar con Python.

[5]El término conteo referencial se centra en el algoritmo de colección de basura en memoria implementado en Python.

- Python no es un lenguaje con ejecución rápido como los lenguajes compilados, pero sin embargo es más flexible y portable, no obstante, posee muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi-interpretado.
- En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la PSF[6]. La última versión estable del lenguaje es la 3.1.1.

1.7.2. GTK+

Características:

- GTK+[7] es un grupo importante de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario, fundamentalmente para el entorno gráfico GNOME[8].
- Fue creado para desarrollar el programa manipulador de imágenes GIMP[9], sin embargo actualmente es muy usada por muchos otros programas en los sistemas GNU/Linux.
- Es una librería libre bajo los términos de LGPL[10] y es parte del proyecto GNU.
- Se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Perl, PHP y Python.
- GTK+ lleva a cabo la comunicación entre objetos usando los llamados callbacks[11].

1.7.3. PyGTK

Es un binding de la biblioteca gráfica GTK para el lenguaje de programación Python, que permite crear fácilmente programas con una interfaz gráfica de usuario. Además de su facilidad de uso y la creación rápida de prototipos, tiene la capacidad para lidiar con texto en

[6]Del inglés Python Software Foundation
[7]Del inglés The GIMP Toolkit.
[8]Acrónimo del inglés GNU Network Object Model Environment.
[9]Del inglés GNU Image Manipulation Program.
[10]Del inglés Lesser General Public License
[11]Función que recibe como argumento la dirección o puntero de otra función.

varios idiomas para aplicaciones complejas. Las aplicaciones PyGTK son multiplataforma y capaces de ejecutarse, sin modificarse, en Linux, Windows y otras plataformas.

1.7.4. GtkSourceView

Librería C portable que amplía el marco estándar de GTK+ para la edición de texto con soporte para resaltado de sintaxis configurable, ilimitado deshacer/rehacer, compatible con UTF-8, impresión y otras características típicas de un editor de código fuente. Forma parte del entorno de escritorio GNOME y está bajo la licencia GNU LGPL 2.1.

1.7.5. PyGtkSourceView

Módulo que permite desde Python utilizar la librería GtkSourceView. Brinda todo el poder de GtkSourceView con una interfaz muy familiar para los programadores de Python, incluidos los usuarios de la biblioteca de PyGTK. Licenciado bajo la licencia GNU LGPL 2.1.

1.7.6. Eclipse

Eclipse IDE está considerado como uno de los mejores entornos de programación en la actualidad. Desarrollado por la Fundación Eclipse, una organización sin ánimos de lucro que publica oficialmente todos los proyectos relacionados con Eclipse y su objetivo principal es el desarrollo de una plataforma libre de desarrollo que contenga todas las herramientas necesarias para el ciclo completo de un software determinado[17]. Licenciado bajo la Licencia Pública de Eclipse. Actualmente la versión estable es la 3.5 con nombre de código: Galileo.

1.7.7. PyDEV

PyDev es un IDE de Python para Eclipse que tiene un conjunto de ventajas como son:

- Completado de código.
- Resaltado de sintaxis.

- Análisis de código.
- Depurador.
- Consola interactiva.

1.8. Conclusiones

En el capítulo recién concluido se abordaron algunos conceptos relacionados con el objeto de investigación, como es el caso de editor de texto, editor de código y depurador, con el fin de ubicar al lector en el tema a tratar. Se analizaron los principales editores de código que pueden ser utilizados en la gestión de los programas de Octave. Ello permitió concluir que es necesario el desarrollo de una herramienta que permita la edición y depuración de código Octave desde el EIDMAT; se realizó la selección de la metodología que guiará el proceso de desarrollo de la aplicación. Además se documentaron las tecnologías y lenguaje a emplear.

Capítulo 2

Planificación y desarrollo de la propuesta de solución.

En este capítulo se inicia el desarrollo de la solución propuesta guiado por la metodología SXP. Se presentan los principales artefactos generados en las primeras fases. Donde se definen un conjunto de actividades de las que se obtienen los documentos relacionados con la concepción inicial del sistema, los requisitos, el diseño, las tareas a realizar durante la implementación y la obtención del código fuente del software.

2.1. Concepción del sistema

Producto de los primeros encuentros con el cliente se genera la Plantilla de Concepción del Sistema, donde se refleja una visión general del producto a implementar. Esta incluye un conjunto de informaciones muy valiosas que dan paso al inicio de la solución propuesta, como es el caso de la descripción del sistema, o sea, ¿Qué es lo que se va a desarrollar?. En esta también se enuncian los roles que van a existir en el proyecto y sus responsabilidades.

2.1.1. Descripción de la propuesta de solución

Como se ha abordado hasta este punto de la investigación, para solucionar el problema tratado se propone el desarrollo de un módulo integrado a la plataforma EIDMAT, que agilice el proceso de edición y depuración de código Octave sobre dicho entorno (Ver Figura 2.1). El editor depurador propuesto, a modo general, debe proveer al usuario un conjunto de funcionalidades, las cuales se listan a continuación:

- Proveer las funcionalidades básicas de un editor de texto (copiar, cortar, pegar, abrir, guardar, eliminar, deshacer, rehacer, seleccionar).
- Resaltar en el código escrito las funciones y palabras reservadas del lenguaje.
- Completado del código.
- Ejecución de archivos ".m"
- Depuración del código haciendo uso de puntos de ruptura y la ejecución paso a paso.
- Plegado del código.

La interfaz principal del MED estará compuesta fundamentalmente por los elementos siguientes:

- Una barra de menú, en la que figurarán los menús descolgables de la aplicación (File, Edit, Window, Help, Debug) que mostrarán todas las opciones que pueden ser ejecutadas sobre la herramienta (New, Open, Save, Delete, Select All, Run).
- La barra de herramientas, compuesta por botones para los comandos más usados (New M-File, Open, Save, Copy, Run), puesto que los iconos hacen más fácil saber que es lo que hace cada botón.
- La vista de texto, en la que el usuario podrá escribir el código del programa, en esta será posible la exploración por pestañas. Además se podrá acceder a los comandos más comunes mediante un menú contextual que se mostrará al presionar click derecho.

- Una barra de estado, que mostrará un conjunto de informaciones útiles al usuario, como es el número de línea y de columna donde se encuentra posicionado el cursor, así como la dirección donde se encuentra guardado el documento.

Otro aspecto importante es que, al ser una aplicación integrada al EIDMAT posibilita al usuario hacer uso de todas las prestaciones hoy disponibles en este último. Además la interacción con el motor de cálculo Octave se realiza a través de este entorno.

Para la implementación del editor depurador se siguen básicamente los pasos siguientes:

- Desarrollar un editor de texto que permita al usuario realizar todas las funcionalidades básicas para la edición de documentos (cortar, copiar, seleccionar, pegar, eliminar, guardar, abrir).
- Añadir al editor de texto un conjunto de funcionalidades que faciliten la edición de código fuente (resaltado de palabras reservadas, completado de código, pareo de llaves, plegado e indentado automático del código).
- Agregar al editor de código funcionalidades de depurador, o sea, que permita la localización y reparación de errores en el código.

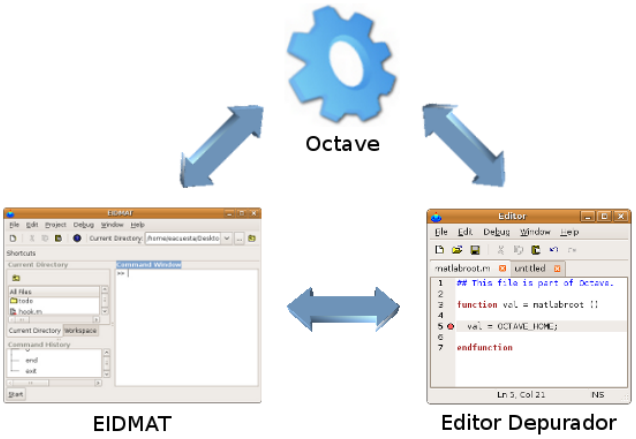


Figura 2.1: Propuesta de solución.

2.1.2. Planificación del proyecto por roles

Rol	Responsabilidad	Nombre
Gerente	Responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el proyecto	Eduardo Cuesta
Cliente	Participa en todas las tareas relacionadas con los requisitos del software	Msc. Alexeis Companioni
Programadores	Definir las tareas de ingeniería y producir el código fuente del sistema	Yurenia Hernández Eduardo Cuesta
Analista	Elabora la concepción del sistema, las historias de usuario y las pruebas funcionales. Trabaja en colaboración con el cliente.	Yurenia Hernández
Diseñador	Responsable del diseño del sistema, así como de los prototipos de interface, máximo responsable de la realización del diseño de las metáforas.	Eduardo Cuesta
Encargado de Pruebas	Ejecuta las pruebas regularmente y difunde los resultados en el equipo	Msc. Alexeis Companioni

Tabla 2.1: Planificación del proyecto por roles.

2.2. Modelo de dominio

Un modelo del domino o también denominado modelo conceptual captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las "cosas" que existen, o los eventos que suceden en el entorno que trabaja el sistema. Muchos de los objetos del dominio o clases pueden obtenerse de una especificación de requisitos, o mediante la entrevista con los expertos del dominio.

Se crea con el objetivo de documentar los conceptos dominantes y el vocabulario del sistema, además proporciona una vista estructural del mismo. Es utilizado en un nivel bajo en el ciclo de desarrollo de software, ya que la semántica demostrada se puede utilizar en el código fuente.

Seguidamente se muestra el modelo de dominio de la propuesta:

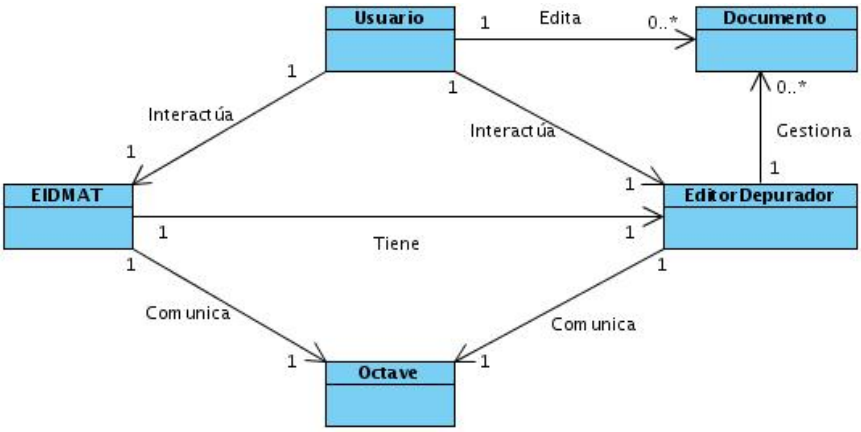


Figura 2.2: Modelo de dominio de la propuesta de solución.

2.3. Captura de requisitos

Una vez que se cuenta con una vista conceptual del sistema que se pretende desarrollar, el próximo paso consiste en la captura de los requisitos asociados a la propuesta; para esto, se parte de un encuentro con los clientes a partir de donde se genera la Lista de Reserva del Producto (LRP). Esta lista recoge la totalidad de las exigencias inherentes al desarrollo y constituye además la base para la confección de las historias de usuario relativas a cada uno de los requisitos según su prioridad.

2.3.1. Lista de Reserva del Producto

La Lista de Reserva del Producto es el primer artefacto generado en la etapa de Captura de requisitos de la metodología SXP, es una lista priorizada que define el trabajo que se va a realizar en el proyecto. Cuando un proyecto comienza es muy difícil tener claro todos los requerimientos sobre el producto. Sin embargo, suelen surgir los más importantes que casi siempre son más que suficientes para un Sprint (Iteración)[11].

A continuación se muestra la LRP de la solución propuesta:

Prioridad	**Ítem ***	**Descripción**	**Estimación (semanas)**	**Estimado por**
Muy Alta				
	1	Crear documento.	$\frac{1}{2}$	ANA. y PROG.
	2	Abrir documento.	$\frac{1}{2}$	ANA. y PROG.
	3	Guardar documento.	$\frac{1}{2}$	ANA. y PROG.
	4	Cerrar documento.	$\frac{1}{2}$	ANA. y PROG.
	5	Copiar código de un documento.	$\frac{1}{2}$	ANA. y PROG.
	6	Pegar código en un documento.	$\frac{1}{2}$	ANA. y PROG.

	7	Cortar porciones de código de un documento.	$\frac{1}{2}$	ANA. y PROG.
	8	Seleccionar código en el documento.	$\frac{1}{2}$	ANA. y PROG.
	9	Eliminar código escrito en el documento	$\frac{1}{2}$	ANA. y PROG.
Alta				
	1	Crear una vista exploratoria por pestañas de los documentos.	1	ANA. y PROG.
	2	Ejecutar archivos ".m".	2	ANA. y PROG.
	3	Depuración del código Octave.	4	ANA. y PROG.
Media				
	1	Resaltado de la sintaxis del código Octave.	3	ANA. y PROG.
	2	Completado de código Octave.	2	ANA. y PROG.
Baja				
	1	Permitir el plegado de código.	3	ANA. y PROG.
RNF				

	1	Las aplicación bebe poseer una interfaz gráfica amigable y simple, enfocada a los usuarios finales.		
	2	El sistema deberá funcionar sobre plataforma GNU/Linux.		
	3	Tener instalada la versión de Octave 3.2.3.		
	4	Tener instalado el módulo pygtksourceview2 $\geq$ 2.9.2.		
	5	El sistema debe ser extensible permitiendo agregar nuevas funcionalidades.		

Tabla 2.2: Lista de Reserva del Producto.

2.3.2. Historias de usuario y prototipos de interfaz de usuario

Las historias de usuario son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en

unas semanas[18].

Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores, para ser implementadas durante una iteración. El formato de tarjeta es muy provechoso a la hora de realizar pruebas de aceptación.

A continuación se exponen las historias de usuarios correspondientes al sistema a desarrollar, así como las interfaces relacionadas con estas. A pesar de que el cliente tiene un amplio dominio de las características del sistema, es válido destacar que es sólo una planificación inicial la cual puede cambiar para ir adecuándola a sus necesidades y nuevas propuestas.

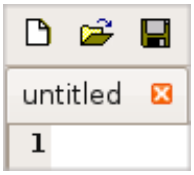
Historia de Usuario	
Número: HU-1	**Nombre Historia de Usuario:** Gestionar Documento
Modificación de Historia de Usuario Número: 0	
Usuario: Yurenia Hernández	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 2 semanas
Descripción: Esta sección garantiza crear, abrir, guardar y cerrar un documento.	
Prototipo de interfase: 	

Tabla 2.3: Historia de usuario Gestionar Documento.

Historia de Usuario	
Número: HU-2	**Nombre Historia de Usuario:** Editar Documento
Modificación de Historia de Usuario Número: 0	
Usuario: Eduardo Cuesta	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Bajo	**Puntos Reales:** 2 semanas
Descripción: Esta sección garantiza las opciones de edición que son posible realizar sobre un documento(copiar, cortar, pegar, eliminar, seleccionar, deshacer, rehacer).	
Observaciones: Todas las opciones de edición se realizan sobre el documento que se encuentre activo.	
Prototipo de interfase: 	

Tabla 2.4: Historia de usuario Editar Documento.

Historia de Usuario	
Número: HU-3	**Nombre Historia de Usuario:** Crear Vista Exploratoria por Pestañas de los Documentos
Modificación de Historia de Usuario Número: 0	
Usuario: Yurenia Hernández	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1 semanas
Riesgo en Desarrollo: Bajo	**Puntos Reales:** $\frac{1}{2}$ semanas
Descripción: Se muestran los documentos abiertos en un sistema de pestañas en la ventana principal de la aplicación.	
Observaciones: Permite al usuario abrir más de un documento en la misma ventana.	

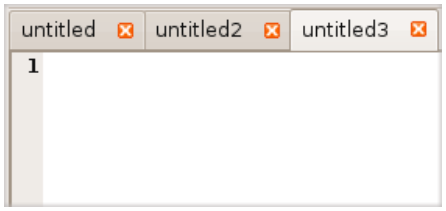
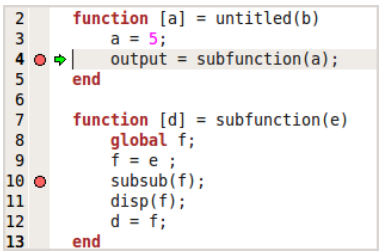
Prototipo de interfase: 

Tabla 2.5: Historia de usuario Crear Vista Exploratoria por Pestañas.

Historia de Usuario	
Número: HU-4	**Nombre Historia de Usuario:** Ejecutar Archivos ".m"
Modificación de Historia de Usuario Número: 0	
Usuario: Yurenia Hernández	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 1 semanas
Descripción: Esta sección garantiza la ejecución de los archivos de código Octave(.m) abiertos en la aplicación, cuya salida se mostrará en la ventana de comandos de EIDMAT.	
Prototipo de interfase: 	

Tabla 2.6: Historia de usuario Ejecutar Archivos ".m".

Historia de Usuario	
Número: HU-5	**Nombre Historia de Usuario:** Depuración del Código Octave
Modificación de Historia de Usuario Número: 0	
Usuario: Eduardo Cuesta	**Iteración Asignada:** 2
Prioridad en Negocio: Alta	**Puntos Estimados:** 4 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 4 semanas
Descripción: Permite la localización y reparación de errores mediante el uso de puntos de ruptura y la ejecución paso a paso.	
Observaciones: Se muestra una marca (flecha) en la línea donde está detenida la ejecución. Además se visualizan los puntos de ruptura que están presentes en el documento.	
Prototipo de interfase: 	

```
2     function [a] = untitled(b)
3         a = 5;
4         output = subfunction(a);
5     end
6
7     function [d] = subfunction(e)
8         global f;
9         f = e ;
10        subsub(f);
11        disp(f);
12        d = f;
13    end
```

Tabla 2.7: Historia de usuario Depuración del Código Octave.

Historia de Usuario	
Número: HU-6	**Nombre Historia de Usuario:** Resaltado de la Sintaxis del Código Octave
Modificación de Historia de Usuario Número: 0	
Usuario: Eduardo Cuesta	**Iteración Asignada:** 3

Prioridad en Negocio: Media	**Puntos Estimados:** 3 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 semanas
Descripción: Esta sección garantiza el resaltado de las funciones y palabras reservadas del código Octave.	
Prototipo de interfase:	

```
function Y = ind2gray (X, map)

  if (nargin < 1 || nargin > 2)
    print_usage ();
  elseif (nargin == 1)
    map = colormap ();
  endif

  [rows, cols] = size (X);

  ## Convert colormap to intensity values (the first column of the
  ## result of the call to rgb2ntsc) and then replace indices in
  ## the input matrix with indexed values in the output matrix (indexed
  ## values are the result of indexing the intensity values by the
  ## elements of X(:)).

  Y = reshape (((rgb2ntsc (map))(:,1))(X(:)), rows, cols);

endfunction
```

Tabla 2.8: Historia de usuario Resaltado de la Sintaxis del Código Octave.

Historia de Usuario	
Número: HU-7	**Nombre Historia de Usuario:** Completado de Código Octave
Modificación de Historia de Usuario Número: 0	
Usuario: Yurenia Hernández	**Iteración Asignada:** 3
Prioridad en Negocio: Media	**Puntos Estimados:** 2 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 semanas
Descripción: Esta sección garantiza el completado de las funciones y palabras reservadas en el código Octave.	
Prototipo de interfase:	

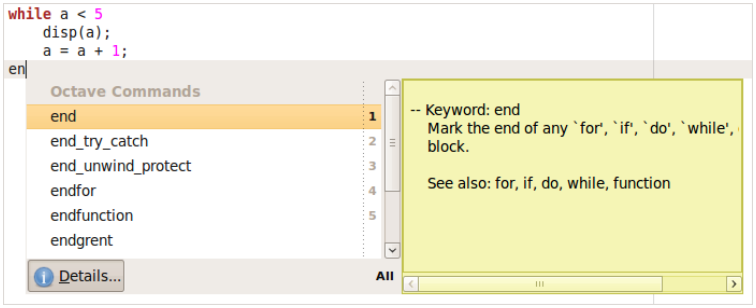


Tabla 2.9: Historia de usuario Completado de Código Octave.

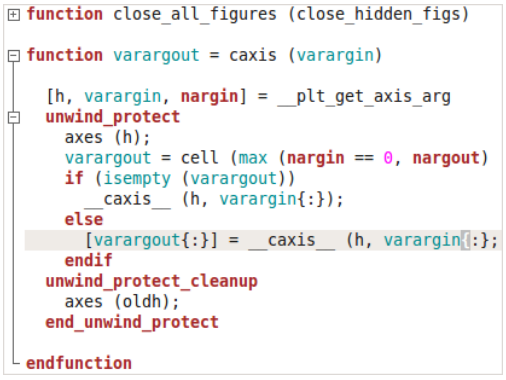
Historia de Usuario	
Número: HU-8	**Nombre Historia de Usuario:** Plegado de Código
Modificación de Historia de Usuario Número: 0	
Usuario: Eduardo Cuesta	**Iteración Asignada:** 3
Prioridad en Negocio: Baja	**Puntos Estimados:** 3 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 2 semanas
Descripción: Esta sección garantiza el plegado de código, posibilitando agrupar el código Octave por las estructuras de bloque definidas en el documento.	
Prototipo de interfase: 	

Tabla 2.10: Historia de usuario Plegado de Código.

2.4. Lista de riesgos

Durante todo el ciclo de desarrollo de un software existen un conjunto de riesgos que en caso de materializarse podrían comprometer el desarrollo, por lo que se hace imprescindible la gestión de los mismos.

SXP propone la creación de la Plantilla Lista de Riesgos, en la que quedan identificados los posibles riesgos que pueden incidir sobre el desarrollo del software, así como las estrategias trazadas para mitigarlos. A pesar de ser imposible definir desde un inicio la totalidad de los riesgos asociados al desarrollo de un proyecto siempre se tendrán en cuenta la mayoría de estos, disminuyéndose así lo más posible las vulnerabilidades en el proceso. Esta plantilla propicia algunas ventajas, tales como[19]:

- Se definen los posibles riesgos, así como la forma de mitigarlos, lo que disminuye el efecto de los mismos, si ocurrieran.
- Se lleva un control de todos los problemas que han azotado al proyecto, así como de la manera que fueron enfrentados y el impacto que tuvieron en el proceso de desarrollo.

Los posibles riesgos durante el proceso de desarrollo del MED se pueden encontrar en la Plantilla Lista de Riesgos (Ver Anexo 6).

2.5. Diseño con metáforas

Una de las mejores prácticas que selecciona SXP de la metodología XP, es la definición del sistema mediante una metáfora o conjunto de metáforas compartidas por el cliente y el equipo de desarrollo, acerca de cómo debería funcionar el sistema. Para la definición de la metáfora del sistema a desarrollar es necesario tener claro que se entiende por este término.

¿Qué es una metáfora?

Una metáfora para el sistema es una historia que todo el mundo puede contar acerca de cómo el sistema funciona. Su objetivo es proporcionar a todo el equipo una misma visión del fin del sistema y de su arquitectura general. Con ello se facilita que todos los desarrolladores hablen un mismo idioma y que nuevos desarrolladores lo adquieran más rápido y se integren en el proyecto sin dificultades.

En XP no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume de forma evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto, se solventan con la existencia de una metáfora[20].

Plasmar la metáfora por escrito puede suponer, por tanto, una forma de revisar el propio diseño del sistema por parte de los desarrolladores[21].

La metáfora definida para el sistema a desarrollar es:

Mediante el envío de comandos a Octave y la visualización de su estado, la herramienta permitirá la edición y depuración del código de este lenguaje.

Basados en la metáfora definida, se genera la Plantilla del Modelo de Diseño, donde se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto, la cual incluye el diagrama de componentes que se expone a continuación:

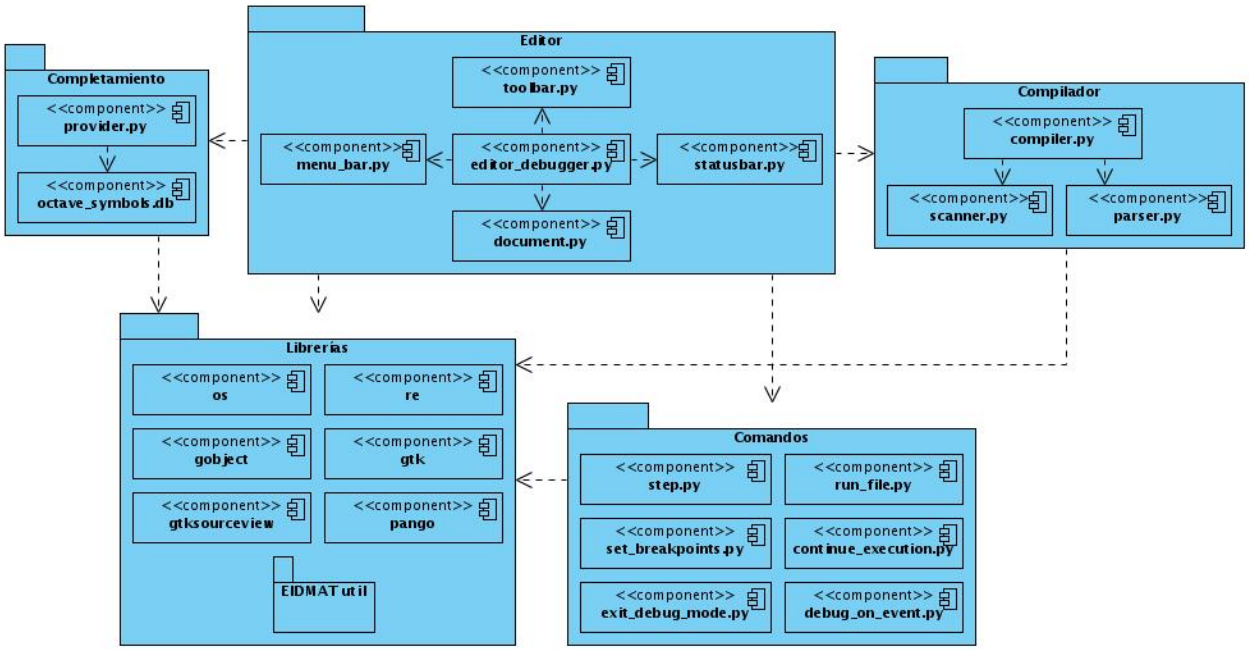


Figura 2.3: Diagrama de componentes del MED.

Descripción de los paquetes que componen el diagrama de componentes:

Paquete Editor: Agrupa aquellos componentes que están relacionados con la interfaz gráfica del editor. El componente principal de este paquete es editor_debugger.py, el cual depende del resto, que no son más que las diferentes partes del editor. Por ejemplo, menu_bar.py representa la barra de menús.

Paquete Completamiento: Contiene los componentes responsables del completado de código en el editor. El componente provider.py obtiene los comandos que serán completados del fichero octave_symbols.db.

Paquete Compilador: Agrupa los componentes encargados de analizar los diferentes archivos de código Octave. El análisis léxico es llevado a cabo por scanner.py; mientras que el análisis sintáctico es realizado por parser.py.

Paquete Comandos: Contiene los componentes que representan los comandos que son enviados a Octave para llevar a cabo la depuración de archivos ".m":

set_breakpoints.py: establece un punto de ruptura.

run_file.py: ejecuta un archivo de código Octave.

continue_execution.py: reanuda la ejecución.

step.py: permite la ejecución paso a paso.

debug_on_event.py: entra en modo depuración en caso de ocurrir un evento determinado.

exit_debug_mode.py: provoca la salida del modo de depuración.

Paquete Librerías: Agrupa las librerías utilizadas por los distintos componentes que conforman el editor depurador.

2.6. Tareas de ingeniería

Seguidamente se definen cada una de las actividades que están asociadas a las historias de usuario. En las que se da a conocer el programador asignado a cada tarea, así como el tiempo necesario para su realización, lo que facilita la estimación del tiempo que abarcará cada historia de usuario en implementarse, de acuerdo a su complejidad.

La HU-1 **"Gestionar Documento"** se divide en cuatro tareas de ingeniería:

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** HU-1
Nombre Tarea: Crear Documento	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 1/01/10	**Fecha Fin:** 4/01/10
Programador Responsable: Yurenia Hernández	
Descripción: Implementar la opción crear nuevo documento	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** HU-1
Nombre Tarea: Abrir Documento	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 5/01/10	**Fecha Fin:** 8/01/10
Programador Responsable: Yurenia Hernández	
Descripción: Implementar la opción abrir documento	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** HU-1
Nombre Tarea: Guardar Documento	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 9/01/10	**Fecha Fin:** 12/01/10
Programador Responsable: Yurenia Hernández	
Descripción: Implementar la opción guardar documento	

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario:** HU-1
Nombre Tarea: Cerrar Documento	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 13/01/10	**Fecha Fin:** 16/01/10
Programador Responsable: Yurenia Hernández	
Descripción: Implementar la opción cerrar documento	

La HU-2 **"Editar Documento"** se divide en una tarea de ingeniería:

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** HU-2
Nombre Tarea: Editar Documento	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 1/01/10	**Fecha Fin:** 16/01/10
Programador Responsable: Eduardo Cuesta	
Descripción: Implementar la opciones de edición: copiar, cortar, pegar, eliminar, seleccionar, deshacer, rehacer	

La HU-3 **"Crear la Vista Exploratoria por Pestañas de los Documentos"** se divide en una tarea de ingeniería:

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** HU-3
Nombre Tarea: Desarrollar la vista exploratoria por pestañas de los documentos	

Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1
Fecha Inicio: 17/01/10	**Fecha Fin:** 24/01/10
Programador Responsable: Yurenia Hernández	
Descripción: Implementar una vista por pestañas de los documentos	

La HU-4 **"Ejecutar archivos ".m""** se divide en dos tareas de ingeniería:

	Tarea de Ingeniería
Número Tarea: 1	**Número Historia de Usuario:** HU-4
Nombre Tarea: Investigar los comandos de Octave que permiten la ejecución de un archivo ".m"	
Tipo de Tarea : Investigación	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 1/02/10	**Fecha Fin:** 4/02/10
Programador Responsable: Yurenia Hernández	
Descripción: Estudiar los comandos de Octave para la ejecución de un archivo de código en este lenguaje	

	Tarea de Ingeniería
Número Tarea: 2	**Número Historia de Usuario:** HU-4
Nombre Tarea: Implementar la ejecución de archivos ".m"	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** $\frac{11}{7}$
Fecha Inicio: 5/02/10	**Fecha Fin:** 15/02/10
Programador Responsable: Yurenia Hernández	
Descripción: Implementar la ejecución de archivos de código Octave(.m)	

La HU-5 **"Depuración del código Octave"** se divide en seis tareas de ingeniería:

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** HU-5
Nombre Tarea: Investigar los comandos de Octave que permiten establecer y eliminar puntos de ruptura en un archivo ".m"	
Tipo de Tarea : Investigación	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 1/02/10	**Fecha Fin:** 4/02/10
Programador Responsable: Eduardo Cuesta	
Descripción: Estudiar los comandos de Octave para establecer y eliminar puntos de ruptura en un archivo ".m"	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** HU-5
Nombre Tarea: Implementar las opciones: establecer y eliminar puntos ruptura	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1
Fecha Inicio: 5/02/10	**Fecha Fin:** 12/02/10
Programador Responsable: Eduardo Cuesta	
Descripción: Permitir poner y eliminar puntos de ruptura en un archivo de código Octave	

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** HU-5

Nombre Tarea: Estudiar los comandos de Octave que permiten la ejecución paso a paso de un archivo ".m"	
Tipo de Tarea : Investigación	**Puntos Estimados:** $\frac{1}{2}$
Fecha Inicio: 13/02/10	**Fecha Fin:** 15/02/10
Programador Responsable: Eduardo Cuesta	
Descripción: Conocer los comandos de Octave que permiten la ejecución paso a paso de un archivo ".m"	

	Tarea de Ingeniería
Número Tarea: 4	**Número Historia de Usuario:** HU-5
Nombre Tarea: Implementar la ejecución paso a paso de un archivo ".m"	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1
Fecha Inicio: 16/02/10	**Fecha Fin:** 21/02/10
Programador Responsable: Eduardo Cuesta	
Descripción: Permitir la ejecución paso a paso de un archivo ".m"	

	Tarea de Ingeniería
Número Tarea: 5	**Número Historia de Usuario:** HU-5
Nombre Tarea: Visualizar una marca donde está detenida la ejecución	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1
Fecha Inicio: 21/02/10	**Fecha Fin:** 25/02/10
Programador Responsable: Eduardo Cuesta	
Descripción: Se muestra una flecha donde está detenida la ejecución de un archivo de código Octave en todo momento	

Tarea de Ingeniería	
Número Tarea: 6	**Número Historia de Usuario:** HU-5
Nombre Tarea: Implementar operación: salir del modo de depuración	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 1
Fecha Inicio: 25/02/10	**Fecha Fin:** 28/02/10
Programador Responsable: Eduardo Cuesta	
Descripción: Se desarrolla la opción salir del modo depuración, la que cuando es invocada detiene la ejecución del archivo	

La HU-6 **"Resaltado de la Sintaxis del Código Octave"** se divide en dos tareas de ingeniería:

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** HU-6
Nombre Tarea: Estudiar las funciones y palabras reservadas de Octave	
Tipo de Tarea : Investigación	**Puntos Estimados:** 1
Fecha Inicio: 1/03/10	**Fecha Fin:** 7/03/10
Programador Responsable: Eduardo Cuesta	
Descripción: Conocer funciones y palabras reservada del lenguaje Octave	

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario:** HU-6

Nombre Tarea: Implementar el resaltado de la sintaxis del Código Octave	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 8/03/10	**Fecha Fin:** 20/03/10
Programador Responsable: Eduardo Cuesta	
Descripción: Garantiza el resaltado las funciones y palabras reservadas del lenguaje Octave	

La HU-7 **"Completado de Código Octave"** se divide en una tarea de ingeniería:

	Tarea de Ingeniería
Número Tarea: 1	**Número Historia de Usuario:** HU-7
Nombre Tarea: Implementar el completado de código Octave	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 1/03/10	**Fecha Fin:** 20/03/10
Programador Responsable: Yurenia Hernández	
Descripción: Garantiza el completado de las funciones y palabras reservadas en los archivos de código Octave	

La HU-8 **"Plegado de Código"** se divide en dos tareas de ingeniería:

	Tarea de Ingeniería
Número Tarea: 1	**Número Historia de Usuario:** HU-8
Nombre Tarea: Investigar cuales son las estructuras de bloque del lenguaje Octave	

Tipo de Tarea : Investigación	**Puntos Estimados:** 2
Fecha Inicio: 21/03/10	**Fecha Fin:** 4/04/10
Programador Responsable: Eduardo Cuesta	
Descripción: Dominio de las estructuras de bloque que define el lenguaje Octave	

	Tarea de Ingeniería
Número Tarea: 2	**Número Historia de Usuario:** HU-8
Nombre Tarea: Implementar el plegado de código	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 2
Fecha Inicio: 5/04/10	**Fecha Fin:** 20/04/10
Programador Responsable: Eduardo Cuesta	
Descripción: Garantiza la agrupación del código Octave por estructuras de bloque	

2.7. Plan de liberación

Planificar acertadamente el proyecto es una forma de estimar el tiempo necesario para su desarrollo. De aquí que SXP propone la elaboración del Plan de Liberación, donde el cliente decide qué historias de usuario deben ser incluidas en un lanzamiento. Las historias de mayor riesgo y mayor prioridad se incluyen en las primeras iteraciones. La elaboración de este artefacto proporciona ventajas tales como:

- Determina las historias de usuario más importantes, y las ubican en las iteraciones según su prioridad.

- Divide el proceso de desarrollo de software en iteraciones, proyectando el trabajo a realizar en cada una de ellas.
- Define las entregas intermedias y final del software a desarrollar.

A continuación se muestra el plan de liberación que guiará el desarrollo del producto:

Release	**Descripción de la iteración**	**Orden de la HU a implementar**	**Duración total**
1	Al concluir la iteración el usuario tendrá la posibilidad de gestionar los documentos, así como, la edición de estos, los que se mostrarán en una vista exploratoria por pestañas	HU-1 HU-2 HU-3	1/01/10 - 24/01/10
2	En esta iteración se incorpora al editor la opción de ejecutar un archivo de código Octave; y su depuración, haciendo uso de puntos de ruptura y la ejecución paso a paso	HU-4 HU-5	1/02/10 - 28/02/10
3	En esta iteración se le garantiza al usuario el resaltado de las funciones y palabras reservadas de Octave, así como su completado. Además de añadir la posibilidad de plegar el código	HU-6 HU-7 HU-8	1/03/10 - 20/04/10

Tabla 2.30: Plan de liberación.

2.8. Conclusiones

En este capítulo se realizó la descripción de la propuesta de solución como factor fundamental para la obtención de un producto que cumpla las expectativas de clientes y desarrolladores. Se capturaron los requisitos del software definidos por el cliente. También se realizó toda la planificación y el diseño del software; y en la misma medida se fue desarrollando el producto.

Capítulo 3

Validación de la solución propuesta.

Una parte fundamental en el ciclo de desarrollo del software es la etapa de pruebas. Proceso que permite verificar y revelar la calidad de un producto.

En este capítulo se depura un archivo de código Octave, haciendo uso de la nueva propuesta. Además se presentan los casos de pruebas de aceptación realizados a la aplicación en cada una de las iteraciones, como hito fundamental para continuar a la próxima iteración. De manera colateral se muestran las funcionalidades que hasta la fecha implementa el sistema.

3.1. Solución de un ejercicio práctico

A continuación se muestran los pasos necesarios para depurar un archivo de código Octave, con el objetivo de mostrar la facilidad y agilidad a la hora realizar este proceso con el editor depurador desarrollado. Para ello se da solución al ejercicio **"Cálculo de una integral doble"**, tomado del libro **"Introducción informal a Matlab y Octave"** [6].

La integral doble a resolver es $I = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)}\, dx\, dy$, la que tiene como resultado π. El programa desarrollado para solucionar el ejercicio antes expuesto (Ver Anexo 7) se comienza a depurar.

Primeramente se establece un punto de ruptura en la línea 4 y otro en la línea 19, esto se logra dando click sobre la barra de puntos de ruptura en las líneas correspondientes, o posicionando el cursor en la línea deseada y presionando la tecla F12, el resultado se puede observar en la Figura 3.1.

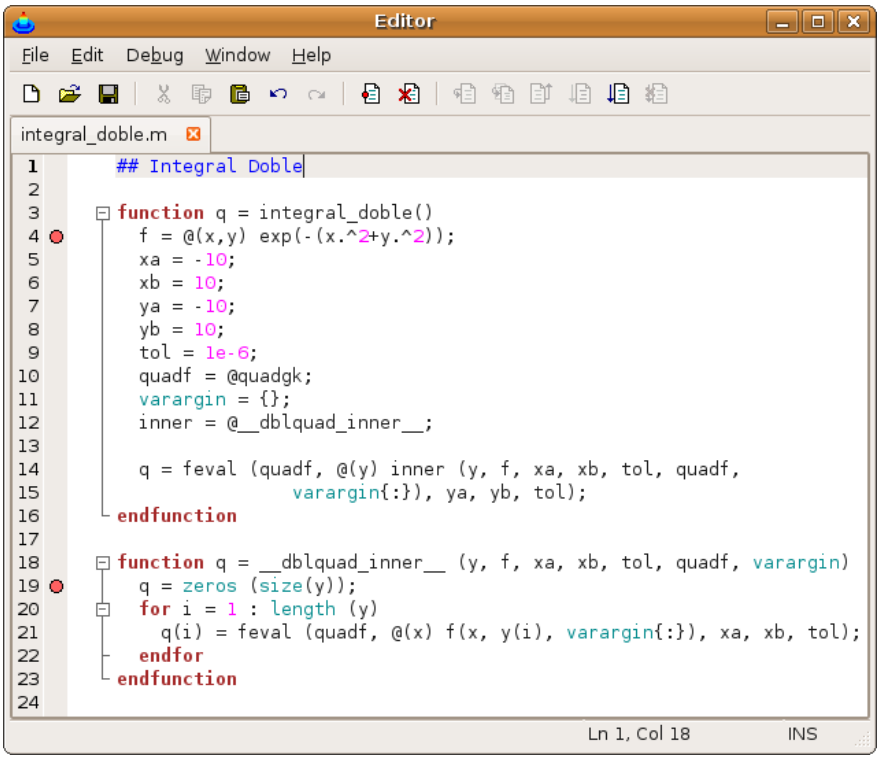


Figura 3.1: Editor con puntos de ruptura.

Seguidamente se ejecuta el archivo presionando la tecla F5. Como se puede observar en la Figura 3.2, se visualiza una marca o flecha verde en la línea 4, lo que indica que la ejecución está detenida en este punto, habilitándose los botones de la barra de herramientas que posibilitan al usuario realizar las distintas operaciones de depuración.

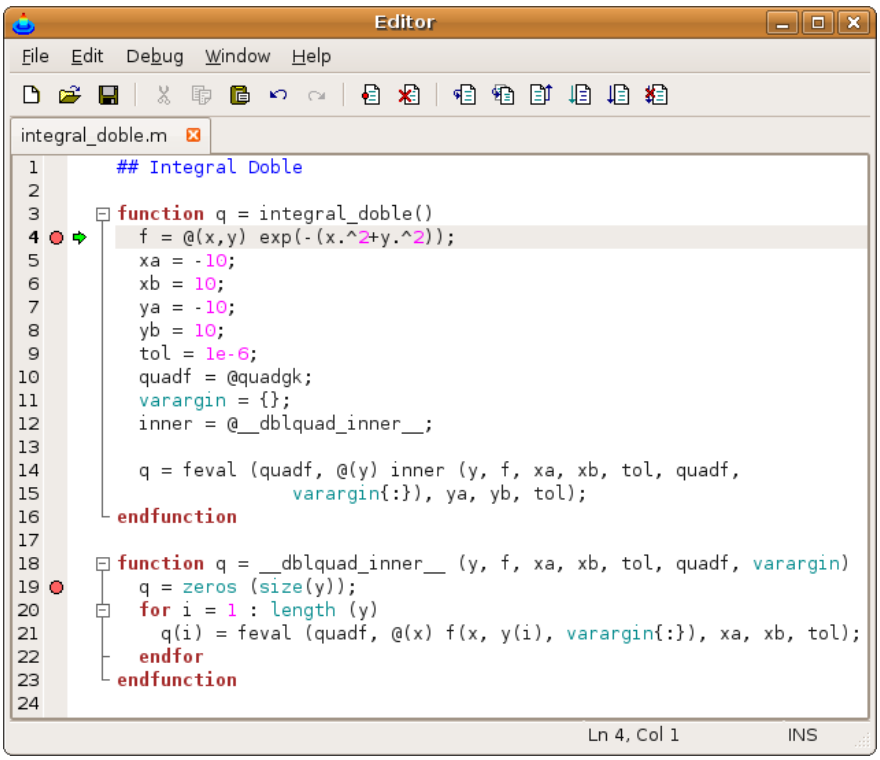


Figura 3.2: Ejecución detenida en un punto de ruptura.

Con el objetivo de ejecutar la instrucción actual y avanzar una línea en la ejecución del programa se presiona la tecla F9 (Ver Figura 3.3).

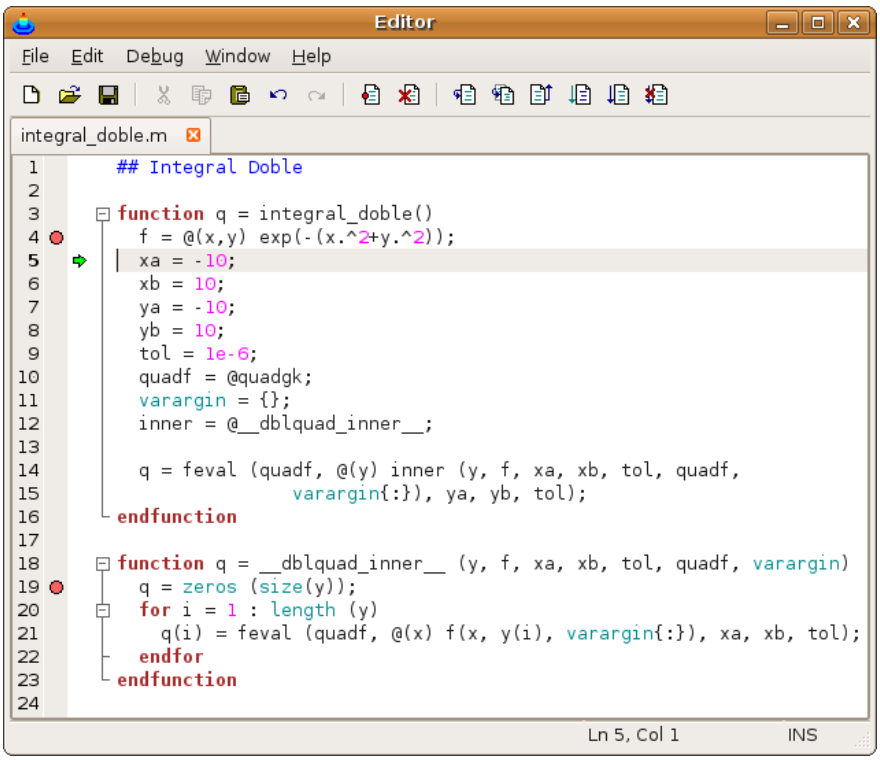


Figura 3.3: Ejecución paso a paso.

Se decide continuar la ejecución del programa hasta el próximo punto de ruptura, para esto se presiona la combinación de teclas Ctrl+F5 (Ver Figura 3.4). En este punto de la ejecución, mediante el Workspace(espacio de trabajo) de EIDMAT, se pueden observar la variables definidas por el usuario hasta el momento, obtener información sobre su tipo, tamaño, así como cambiar sus valores antes de reanudar la ejecución del programa (Ver Figura 3.5).

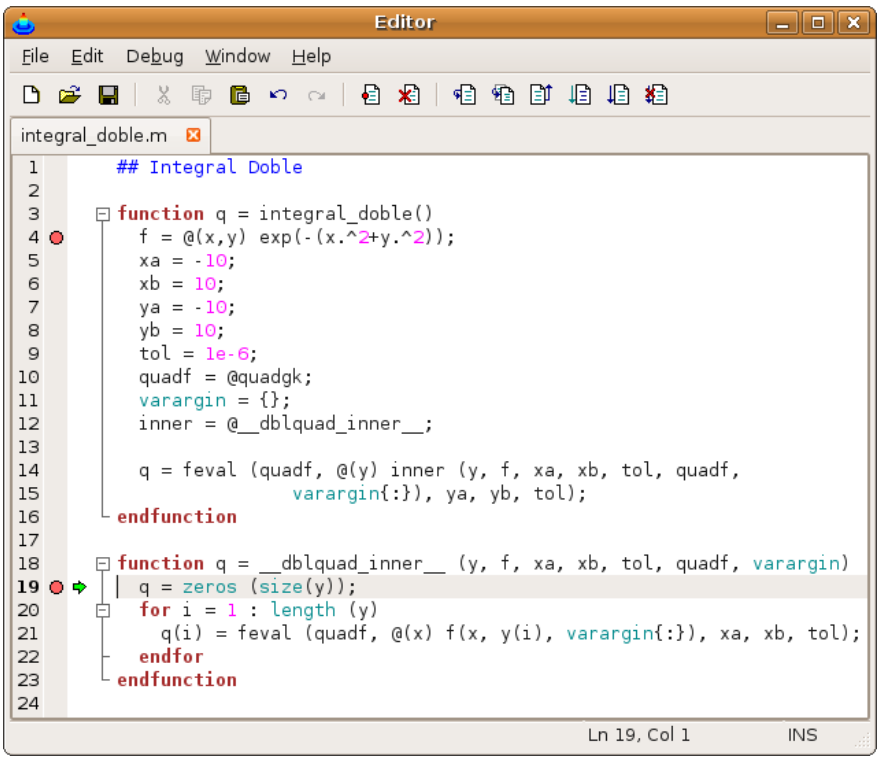


Figura 3.4: Continuación de la ejecución.

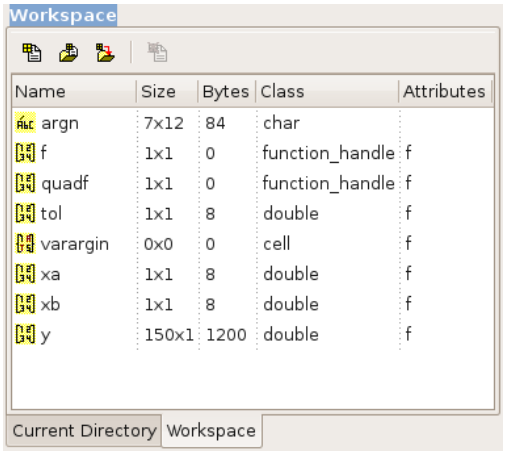


Figura 3.5: Estado de las variables en el Workspace.

Por último se continúa la ejecución hasta el final del programa, mostrándose el resultado del mismo en el CommandWindow(Ventana de comandos) de EIDMAT (Ver Figura 3.6).

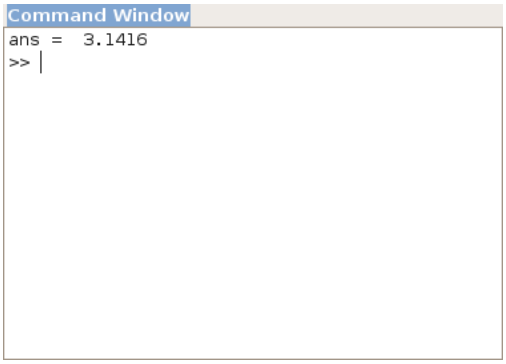


Figura 3.6: Resultado de la ejecución.

3.2. Casos de prueba de aceptación

Durante el desarrollo del sistema se realizaron un conjunto de pruebas de aceptación asociadas a cada una de las historias de usuario, con el objetivo de validar que la propuesta cumple con el funcionamiento esperado, y permitir al usuario del mismo determinar su aceptación. Seguidamente se describe en detalles cada una de estas.

3.2.1. Casos de pruebas para la historia de usuario: HU-1

Esta sección cubre el conjunto de pruebas realizadas a la historia de usuario: **Gestionar Documento**.

Con estas pruebas se pretende comprobar que es posible crear, abrir, guardar y cerrar un documento desde el editor depurador.

	Caso de Prueba de Aceptación
Código Caso de Prueba: MED-HU-1-1	**Nombre Historia de Usuario:** Gestionar Documento
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a la creación de un documento	

Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose
Entrada/Pasos de ejecución: Click sobre la opción de menú File/New/M-File
Resultado Esperado: Se crea un documento
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-1-2	**Nombre Historia de Usuario:** Gestionar Documento
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a abrir un documento	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El archivo que se pretende abrir debe ser de texto plano	
Entrada/Pasos de ejecución: Click sobre la opción de menú File/Open, luego seleccionar el archivo y presionar el botón Open en el cuadro de diálogo “Open File”	
Resultado Esperado: Se abre el documento seleccionado	
Evaluación de la Prueba: Satisfactoria	

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-1-3	**Nombre Historia de Usuario:** Gestionar Documento
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a guardar un documento	

Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose
Entrada/Pasos de ejecución: Click sobre la opción de menú File/-Save As, luego seleccionar el directorio y presionar el botón Save en el cuadro de diálogo "Save file as"
Resultado Esperado: Se guarda el documento
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-1-4	**Nombre Historia de Usuario:** Gestionar Documento
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a cerrar un documento	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El documento a cerrar debe estar abierto en el editor	
Entrada/Pasos de ejecución: Click sobre la "x" que se muestra en la esquina superior izquierda de la pestaña que muestra el documento. El documento se cierra si todos los cambios han sido guardados, sino, se muestra un cuadro de diálogo en el que se selecciona la opción "Close without Saving" o "Save"	
Resultado Esperado: Se cierra el documento	
Evaluación de la Prueba: Satisfactoria	

3.2.2. Caso de prueba para la historia de usuario: HU-2

En esta sección se describe la prueba realizada a la historia de usuario: **Editar Documento**.

Para esta historia se comprueban las opciones de edición, copiar, cortar, pegar, deshacer, rehacer, seleccionar y eliminar sobre un documento.

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-2-1	**Nombre Historia de Usuario:** Editar Documento
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se comprueban las opciones de edición copiar, cortar, pegar, deshacer, rehacer, seleccionar y eliminar sobre un documento	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose	
Entrada/Pasos de ejecución: Se selecciona una porción de texto, se accede a la opción Edit/Copy y luego a la opción Edit/Paste. Seguidamente se selecciona una porción de texto y se accede a la opción Edit/Cut, luego a la opción Edit/Undo y por último a la opción Edit/Redo. Después se accede a la opción Edit/Select All y luego a la opción Edit/Delete	
Resultado Esperado: Se realizan con éxito todas las opciones de edición ejecutadas anteriormente	
Evaluación de la Prueba: Satisfactoria	

3.2.3. Caso de prueba para la historia de usuario: HU-3

En esta sección se describe la prueba realizada a la historia de usuario: **Crear Vista Exploratoria por Pestañas de los Documentos**.

Mediante esta prueba se verifica la posibilidad de abrir más de un documento en la ventana principal de la aplicación.

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-3-1	**Nombre Historia de Usuario:** Crear Vista Exploratoria por Pestañas de los Documentos
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a mostrar más de un documento en la ventana principal de la aplicación	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose	
Entrada/Pasos de ejecución: Se crean dos documentos	
Resultado Esperado: Se muestran los dos documentos en la ventana principal, cada uno en una pestaña	
Evaluación de la Prueba: Satisfactoria	

3.2.4. Caso de prueba para la historia de usuario: HU-4

En esta sección se describe la prueba realizada a la historia de usuario: **Ejecutar Archivos ".m"**.

Para esta historia se comprueba la posibilidad de ejecutar un archivo de código Octave (".m")

desde el editor depurador.

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-4-1	**Nombre Historia de Usuario:** Ejecutar Archivos ".m"
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a la ejecución de un archivo ".m"	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El archivo tiene que ser de código Octave(".m") y estar guardado en disco duro	
Entrada/Pasos de ejecución: Click sobre la opción de menú Debug/Run. El archivo se ejecuta si está en la ruta de búsqueda de Octave, sino, se muestra un cuadro de diálogo en el que se selecciona la opción "Change Directory" o "Add to Path"	
Resultado Esperado: Ejecución del archivo	
Evaluación de la Prueba: Satisfactoria	

3.2.5. Casos de pruebas para la historia de usuario: HU-5

Esta sección cubre el conjunto de pruebas realizadas a la historia de usuario: **Depuración del Código Octave**.

Con estas pruebas se pretende comprobar todo el proceso de depuración de código Octave. Se verifica la posibilidad de establecer y eleminar un punto de ruptura, la visualización de una flecha que indique la línea donde está detenida la ejecución del archivo, así como la capacidad de salir del modo de depuración.

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-5-1	**Nombre Historia de Usuario:** Depuración del código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a establecer un punto de ruptura, donde se detendrá la ejecución	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El archivo tiene que ser de código Octave(".m") y estar guardado en disco duro	
Entrada/Pasos de ejecución: Posicionar el cursor en la línea donde se desea establecer el punto de ruptura, click sobre la opción de menú Debug/Set-Clear Breakpoint. El punto de ruptura se establece si está en la ruta de búsqueda de Octave, sino, se muestra un cuadro de diálogo en el que se selecciona la opción "Change Directory" o "Add to Path"	
Resultado Esperado: Establecimiento del punto de ruptura	
Evaluación de la Prueba: Satisfactoria	

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-5-2	**Nombre Historia de Usuario:** Depuración del código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a eliminar un punto de ruptura que se encuentra establecido en el documento	

Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El archivo tiene que ser de código Octave(".m"). La línea seleccionada tiene que tener un punto de ruptura y estar guardado en disco duro
Entrada/Pasos de ejecución: Posicionar el cursor en la línea donde se desea eliminar el punto de ruptura. Click sobre la opción de menú Debug/Set-Clear Breakpoint
Resultado Esperado: Se elimina el punto de ruptura
Evaluación de la Prueba: Satisfactoria

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-5-3	**Nombre Historia de Usuario:** Depuración del código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se establece un punto de ruptura en el archivo, para comprobar si al ejecutarlo, se muestra una flecha que indique que la ejecución está detenida en esa línea	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. La ejecución del archivo tiene que estar detenida	
Entrada/Pasos de ejecución: Establecer un punto de ruptura en el archivo. Ejecutar el archivo	
Resultado Esperado: Visualización de una flecha en la línea donde se estableció el punto de ruptura	
Evaluación de la Prueba: Satisfactoria	

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-5-4	**Nombre Historia de Usuario:** Depuración del código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se procede a verificar si es posible salir del modo de depuración	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El archivo tiene que estar en modo de depuración	
Entrada/Pasos de ejecución: Click sobre la opción de menú Debug/Exit Debug Mode	
Resultado Esperado: Finalización del modo de depuración	
Evaluación de la Prueba: Satisfactoria	

3.2.6. Caso de prueba para la historia de usuario: HU-6

En esta sección se describe la prueba realizada a la historia de usuario: **Resaltado de la Sintaxis del Código Octave**.

Para esta historia se comprueba el resaltado de las funciones y palabras reservadas de Octave.

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-6-1	**Nombre Historia de Usuario:** Resaltado de la Sintaxis del Código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	

Descripción de la Prueba: Se procede a comprobar el resaltado de las funciones y palabras reservadas de Octave, para ello, se escribe la palabra reservada "while" y la función "round" en un nuevo documento.
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El documento tiene que ser un ".m" o no haber sido guardado nunca
Entrada/Pasos de ejecución: Se crea un documento y se escriben las palabras "while" y "round"
Resultado Esperado: Resaltado de las palabras escritas
Evaluación de la Prueba: Satisfactoria

3.2.7. Caso de prueba para la historia de usuario: HU-7

En esta sección se describe la prueba realizada a la historia de usuario: **Completado de Código Octave**.

Para esta historia se comprueba el completado de las funciones y palabras reservadas de Octave.

	Caso de Prueba de Aceptación
Código Caso de Prueba: MED-HU-7-1	**Nombre Historia de Usuario:** Completado de Código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se escribe la letra "w" con el propósito de ver si se muestran las funciones y palabras reservadas de Octave que comienzan con esta letra, donde sea posible el completado.	

Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El documento tiene que ser un ".m" o no haber sido guardado nunca
Entrada/Pasos de ejecución: Se crea un documento. Se escribe la letra "w" y en la ventana que se muestran las palabras reservadas y funciones de Octave que comienzan con "w" se selecciona la que se desea escribir y se presiona la tecla Enter para el completado.
Resultado Esperado: Completado de la palabra
Evaluación de la Prueba: Satisfactoria

3.2.8. Caso de prueba para la historia de usuario: HU-8

En esta sección se describe la prueba realizada a la historia de usuario: **Plegado de Código**.

Mediante esta prueba se comprueba el plegado del código Octave.

Caso de Prueba de Aceptación	
Código Caso de Prueba: MED-HU-8-1	**Nombre Historia de Usuario:** Completado de Código Octave
Nombre de la persona que realiza la prueba: Alexeis Companioni	
Descripción de la Prueba: Se abre un documento de código Octave, con el objetivo de verificar que es posible el plegado del código en este.	
Condiciones de Ejecución: La aplicación EIDMAT debe estar ejecutándose. El archivo tiene que ser un ".m"	

Entrada/Pasos de ejecución: Se abre un archivo de código Octave. Se posiciona el cursor en la barra de plegado, en la línea que comienza el bloque que se quiere ocultar y se presiona click
Resultado Esperado: Plegado del bloque
Evaluación de la Prueba: Satisfactoria

3.3. Resultados obtenidos

Con el desarrollo de esta aplicación el EIDMAT cuenta con una herramienta que facilita y agiliza todo el proceso de edición y depuración de código Octave. Los usuarios que hagan uso de este entorno integrado podrán realizar dicho proceso de una forma sencilla e intuitiva, sin la necesidad de conocer cada uno de los comandos que hacen posible su realización.

El MED puede adquirirse en su versión 0.1 a través de la sección *Ficheros* de la plataforma GForge. De manera adicional, este se encuentra integrado a la versión 0.3 de EIDMAT la cual sustituirá próximamente la versión 0.2 que está siendo actualmente empleada desde los laboratorios docentes de la facultad 10 de la Universidad de las Ciencias Informáticas.

3.3.1. Acerca de las funcionalidades implementadas

Las principales funcionalidades que posee el producto de software en su primera versión son (Ver Anexos 8, 9 y 10):

- Crear, abrir, guardar y cerrar archivos de texto plano.
- Editar archivos(copiar, cortar, pegar, deshacer, rehacer, seleccionar y eliminar).
- Mostrar los documentos en una vista exploratoria por pestañas.
- Ejecutar archivos de código Octave.

- Depurar el código Octave mediante el uso de puntos de rupturas y la ejecución paso a paso.
- Resaltado de la sintaxis de Octave.
- Completado de código Octave.
- Plegado del código Octave.

3.4. Conclusiones

En este capítulo se mostraron los pasos para llevar a cabo la depuración de un archivo de código Octave, utilizando la aplicación desarrollada; donde se pusieron en práctica cada una de las funcionalidades implementadas, demostrando de esta forma la agilidad y facilidad con que se realiza este proceso. Además se describieron las pruebas que corroboran la calidad del producto; y como resultado de estas se muestran un conjunto de funcionalidades validadas que garantizan la entrega de un producto que responde a las necesidades del cliente.

Conclusiones

Con la realización del presente trabajo se dio cumplimiento a cada uno de los objetivos trazados, pudiéndose destacar de manera general las conclusiones siguientes:

- Se desarrolló un editor depurador de código Octave integrado a la plataforma EIDMAT, que permite la gestión y edición de archivos de código; así como el resaltado de la sintaxis del lenguaje, el completado de las palabras reservadas y la depuración del código.
- Fueron analizados los editores de código existentes que pueden ser utilizados para la edición y depuración de código Octave, determinándose la necesidad de desarrollar una herramienta que se adapte a las características propias del EIDMAT.
- Se validó la funcionalidad del sistema desarrollado a través de un número importante de pruebas de aceptación y mediante la solución de un ejercicio práctico, lo que permitió concluir que las funcionalidades implementadas se encuentran en correspondencia con las necesidades del cliente.

Recomendaciones

En este trabajo se cumplieron todos los requerimientos planteados. No obstante se recomienda:

- Desplegar el producto a gran escala con el objetivo de asegurar su correcto funcionamiento.
- Ampliar el analizador sintáctico realizado de modo que posibilite la detección de errores sintácticos en el código escrito.
- Extender el editor de forma tal que permita la gestión de archivos de código Maxima y R.
- Implementar la integración del editor con algún Sistema de Control de Versiones.

Referencias Bibliográficas

[1] D. E. P. del Toro and L. M. T. Acosta, "*QtOctave: Una Reingeniería por un Desarrollo Genuino*," Junio 2008.

[2] Y. P. Villazon, "*METODOLOGIA PARA LA MIGRACION A SOFTWARE LIBRE DE LAS UNIVERSIDADES DEL MINISTERIO DE EDUCACION SUPERIOR (MES)*," Mayo 2008.

[3] F. S. Foundation, "*La Definición de Software Libre.*" `http://www.gnu.org/philosophy/free-sw.es.html`, 2009. [citado 8 diciembre 2009].

[4] M. C. Juárez, W. G. G. Herrera, and S. T. Sánchez, "*Software libre vs software propietario.*" `http://www.gnu.org/philosophy/free-sw.es.html`, Mayo 2006. [citado 9 diciembre 2009].

[5] D. A. H. Aponte, "*Introducción a GNU Octave.*" `http://nux.ula.ve/manuales/octave/octave.pdf`, Agosto 2007. [citado 21 noviembre 2009].

[6] G. B. Nogueras, "*Introducción informal a Matlab y Octave.*" `http://iimyo.forja.rediris.es/`, Octubre 2008. [citado 6 enero 2010].

[7] J. M. MELIÁN and J. F. H. BALLESTEROS, *La calidad del software y su medida.* No. 84-8004-611-2, CENTRO DE ESTUDIOS RAMON ARECES, S.A., 2003.

[8] A. C. SERVETTO, "*Ambiente Integrado de Ingeniería Automática de Sistemas.*" `http://laboratorios.fi.uba.ar/lsi/p-servetto-proyectodetesis.htm`. [citado 10 enero 2010].

[9] D. P. Roldán, "*Sistema de Clonación y Distribución de Imágenes de Sistemas Operativos*," Junio 2008.

[10] K. Beck, *Extreme Programming Explained*. No. 0201616416, September 1999.

[11] G. M. P. Romero, "*MA-GMPR-UR2 Metodología ágil para proyectos de software libre*," Junio 2008.

[12] A. Calderón and S. Dámaris, "*Metodologías Ágiles*," 2007. Perú.

[13] C. J. S. Álvarez and R. G. F. Díaz, "Metodologías tradicionales vs. metodologías Ágiles," tech. rep., Universidad Técnica Particular de Loja, junio 2007.

[14] H. Kniberg, *SCRUM Y XP DESDE LAS TRINCHERAS*. No. 978-1-4303-2264-1, Enterprise Software Development Series, 2007.

[15] P. S. Foundation, "*HISTORY*." `http://svn.python.org/view/python/trunk/Misc/HISTORY?view=markup&pathrev=51814`. [citado 13 febrero 2010].

[16] P. S. Foundation, "*Python Programming Language*." `http://www.python.org`. [citado 13 febrero 2010].

[17] E. Foundation, "*Explore the Eclipse universe*." `http://www.eclipse.org/`. [citado 15 febrero 2010].

[18] J. H. Canós, P. Letelier, and M. C. Penadés, "*Métodologías Ágiles en el Desarrollo de Software* ." `http://www.willydev.net/descargas/prev/TodoAgil.Pdf`. [citado 11 febrero 2010].

[19] R. F. Céspedes and S. P. García, "*Propuesta de un expediente, para los proyectos productivos del Polo de Software Libre, de la Facultad 10*," Junio 2008.

[20] R. G. Pelegrino, "*Aplicación para el monitoreo de ventanas y notificación de tareas*," Junio 2009.

[21] G. Robles and J. Ferrer, "*Programación eXtrema y Software Libre*," Octubre 2002.

Bibliografía

[1] A. C. SERVETTO, "*Ambiente Integrado de Ingeniería Automática de Sistemas.*" `http://laboratorios.fi.uba.ar/lsi/p-servetto-proyectodetesis.htm.` [citado 10 enero 2010].

[2] A. Calderón and S. Dámaris, "*Metodologías Ágiles,*" 2007. Perú.

[3] C. J. S. Álvarez and R, G. F. Díaz, "*Metodologías Tradicionales vs. Metodologías Ágiles,*" Junio 2008. tech. rep., Universidad Técnica Particular de Loja, junio 2007

[4] D. A. H. Aponte, "*Introducción a GNU Octave.*" `http://nux.ula.ve/manuales/octave/octave.pdf,` Agosto 2007. [citado 21 noviembre 2009].

[5] D. E. P. del Toro and L. M. T. Acosta, "*QtOctave: Una Reingeniería por un Desarrollo Genuino,*" Junio 2008.

[6] D. P. Roldán, "*Sistema de Clonación y Distribución de Imágenes de Sistemas Operativos,*" Junio 2008.

[7] E. Foundation, "*Explore the Eclipse universe.*" `http://www.eclipse.org/.` [citado 15 febrero 2010].

[8] F. S. Foundation, "*La Definición de Software Libre.*" `http://www.gnu.org/philosophy/free-sw.es.html` ,2009. [citado 8 diciembre 2009].

[9] G. B. Nogueras, "*Octave: Una alternativa real a Matlab a coste de cero.*" `http://torroja.dmt.upm.es:81/media/files/paper_logrono.pdf,` 2007. [citado 15 enero 2010].

[10] G. B. Nogueras, "*Introducción informal a Matlab y Octave.*" `http://iimyo.forja.rediris.es/,` Octubre 2008. [citado 6 enero 2010].

[11] G. M. P. Romero, "*MA-GMPR-UR2 Metodología ágil para proyectos de software libre,*" Junio 2008.

[12] G. Robles and J.Ferrer, "*Programación eXtrema y Software Libre,*" Octubre 2002.

[13] H. Kniberg, "*SCRUM Y XP DESDE LAS TRINCHERAS.*" No. 978-1-4303-2264-1, Enterprise Software Development Series, 2007.

[14] J. H. Canós, P. Letelier and M. C. Penadés, "*Métodologías Ágiles en el Desarrollo de Software.*" `http://www.willydev.net/descargas/prev/TodoAgil.Pdf.` [citado 11 febrero 2010].

[15] J. M. Melián and J. F. H. BALLESTEROS, "*La calidad del software y su medida.*" No. 84-8004-611-2, CENTRO DE ESTUDIOS RAMON ARECES, S.A., 2003.

[16] K. Beck, "*Extreme Programming Explained.*" No. 0201616416, September 1999.

[17] M. C. Juárez, W. G. G. Herrera, and S. T. Sánchez, "*Software libre vs software propietario.*" `http://www.gnu.org/philosophy/free-sw.es.html,` Mayo 2006. [citado 9 diciembre 2009].

[18] P. S. Foundation, "*HISTORY.*" `http://svn.python.org/view/python/trunk/Misc/HISTORY?view=markup&pathrev=51814.` [citado 13 febrero 2010].

[19] P. S. Foundation, "*Python Programming Language.*" `http://www.python.org.` [citado 13 febrero 2010].

[20] R. A. H. León and S. C. González, "*EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA,*" Noviembre 2002.

[21] R. F. Céspedes and S. P. García, "*Propuesta de un expediente, para los proyectos productivos del Polo de Software Libre, de la Facultad 10,*" Junio 2008.

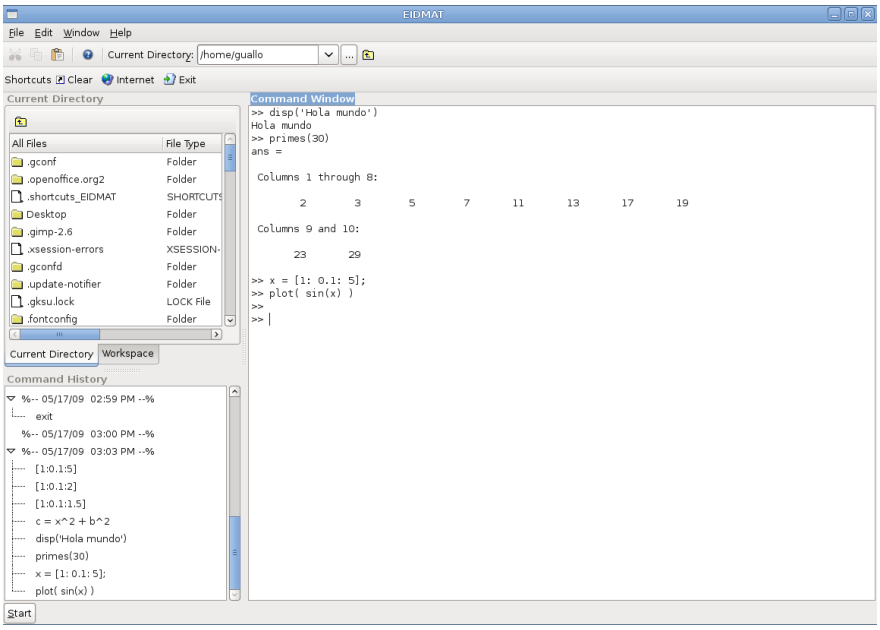
[22] R. G. Duque, "*Python para Todos.*" `http://mundogeek.net/tutorial-python.` [citado 3 diciembre 2009].

[23] R. G. Pelegrino, "*Aplicación para el monitoreo de ventanas y notificación de tareas,*" Junio 2009.

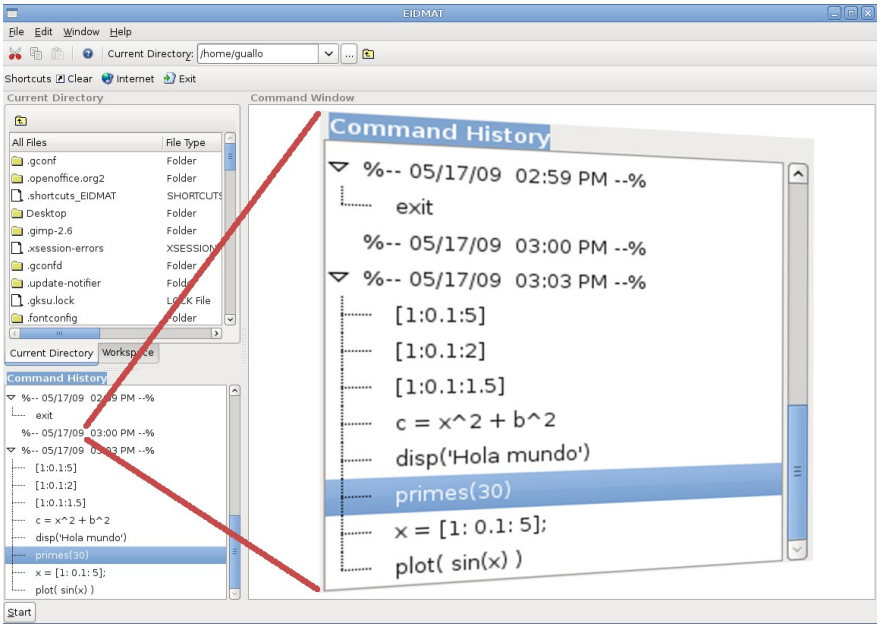
[24] Y. P. Villazon, "*AMETODOLOGIA PARA LA MIGRACION A SOFTWARE LIBRE DE LAS UNIVERSIDADES DEL MINISTERIO DE EDUCACION SUPERIOR (MES),*" Mayo 2008.

Anexos

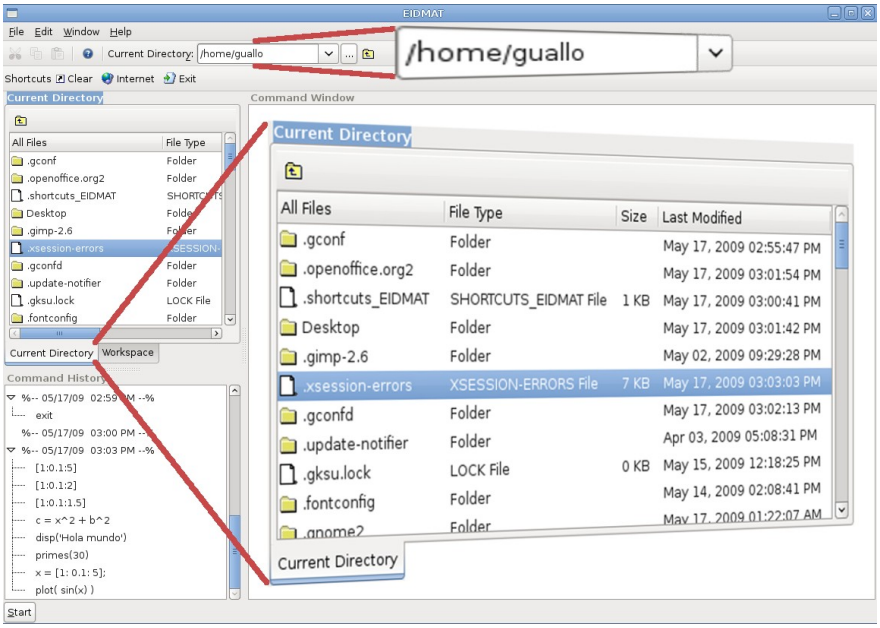
Anexo 1: Módulo principal de EIDMAT



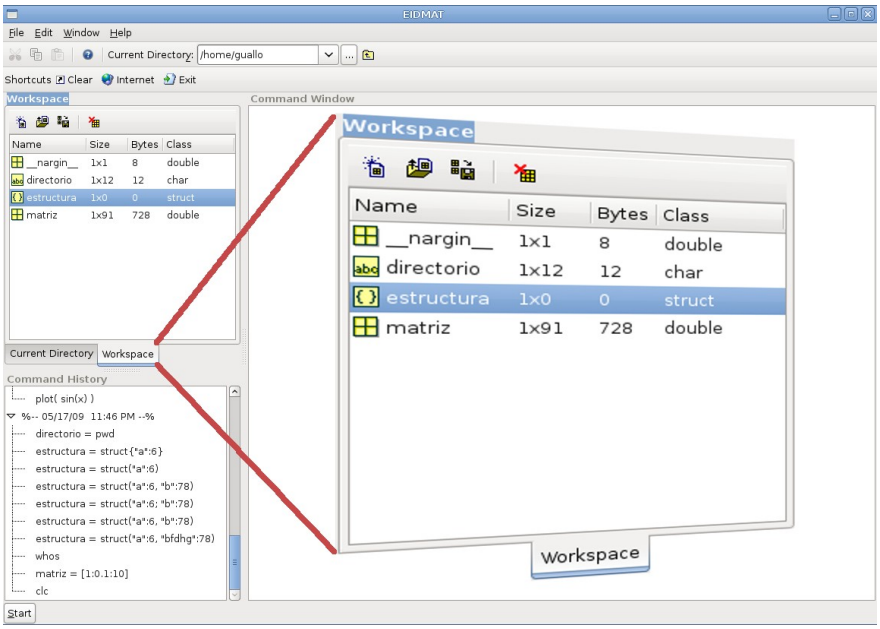
Anexo 2: Historial de comandos de EIDMAT



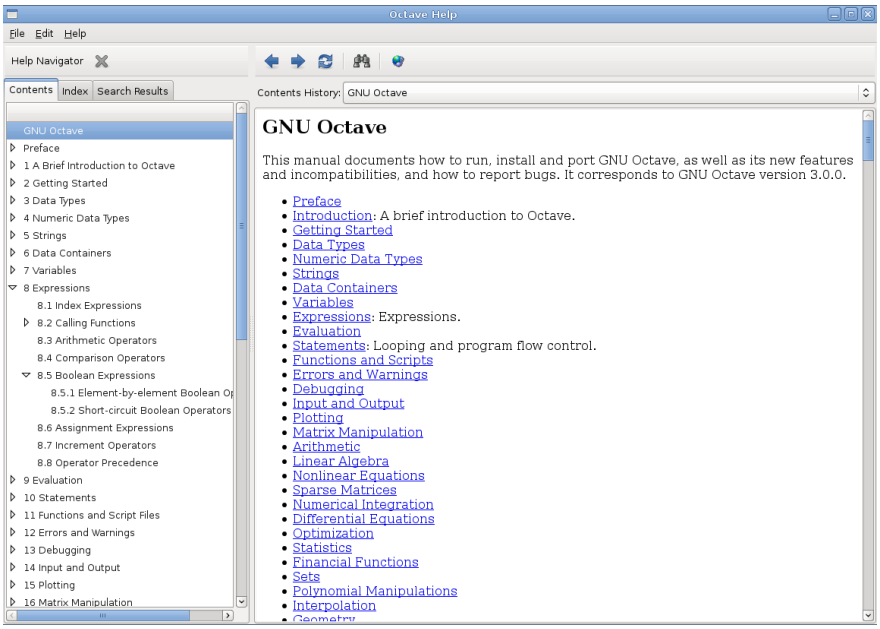
Anexo 3: Ventana de directorio actual de EIDMAT



Anexo 4: Espacio de trabajo de EIDMAT



Anexo 5: Ayuda de EIDMAT



Anexo 6: Lista de riesgos

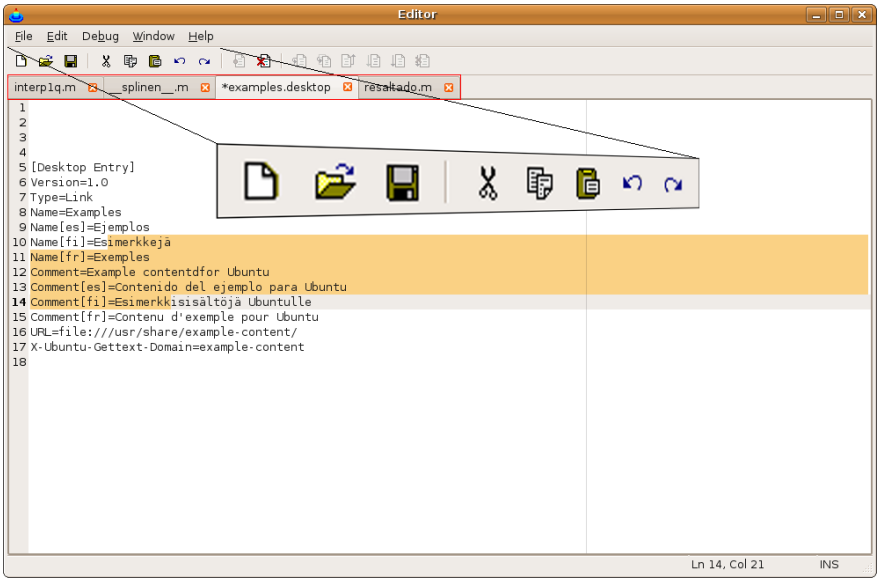
Riesgo	Tipos de Riesgos	Impacto	Descripción	Probabilidad	Efectos	Mitigación de Riesgo
1	Estimación	Desarrollo	Complejidad de las tecnologías a utilizar	Alta	Tolerables	Obteniendo libros específicos del tema
2	Tecnológico	Retraso del trabajo	Rotura de una computadora donde se encuentra información importante	Alta	Serios	Realizando salvas frecuentes de la información
3	Personal	Retraso en las actividades	Enfermedad de uno de los miembros del equipo	Alta	Tolerables	Capacitar varias personas en cada tema
4	Tecnológico	Retraso en las actividades	Afectación del fluido eléctrico	Baja	Tolerables	Buscar horarios alternativos para trabajar

5	Tecnológico	Productividad	Desconexión de la red constantemente	Alta	Serios	Comunicar al jefe de proyecto
6	Tecnológico	Retraso del producto	Carencia de sillas	Media	Serios	Gestionar la asignación de sillas al laboratorio

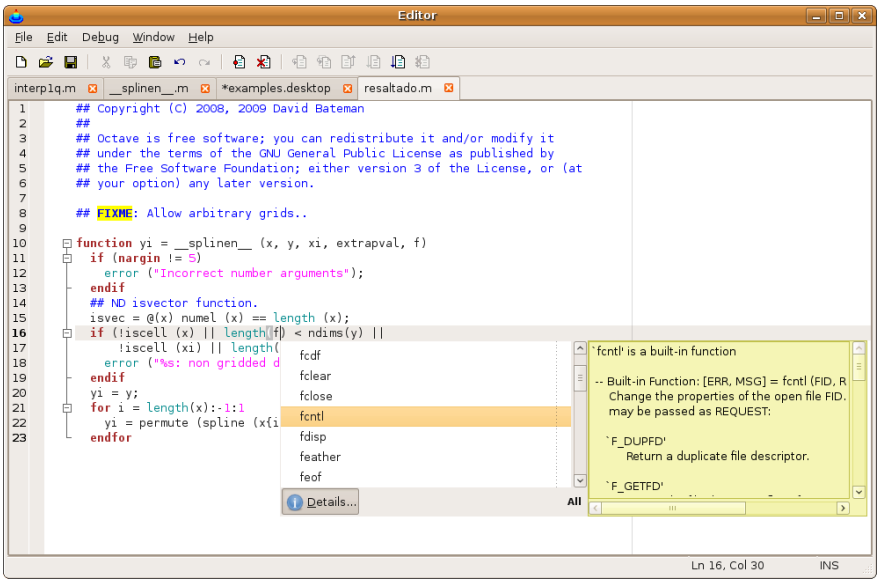
Anexo 7: Programa que soluciona el ejercicio "Cálculo de una integral doble"

```
1  %% Integral Doble
2
3  function q = integral_doble()
4    f = @(x,y) exp(-(x.^2+y.^2));
5    xa = -10;
6    xb = 10;
7    ya = -10;
8    yb = 10;
9    tol = 1e-6;
10   quadf = @quadgk;
11   varargin = {};
12   inner = @__dblquad_inner__;
13
14   q = feval (quadf, @(y) inner (y, f, xa, xb, tol, quadf,
15                  varargin{:}), ya, yb, tol);
16 endfunction
17
18 function q = __dblquad_inner__ (y, f, xa, xb, tol, quadf, varargin)
19   q = zeros (size(y));
20   for i = 1 : length (y)
21     q(i) = feval (quadf, @(x) f(x, y(i), varargin{:}), xa, xb, tol);
22   endfor
23 endfunction
```

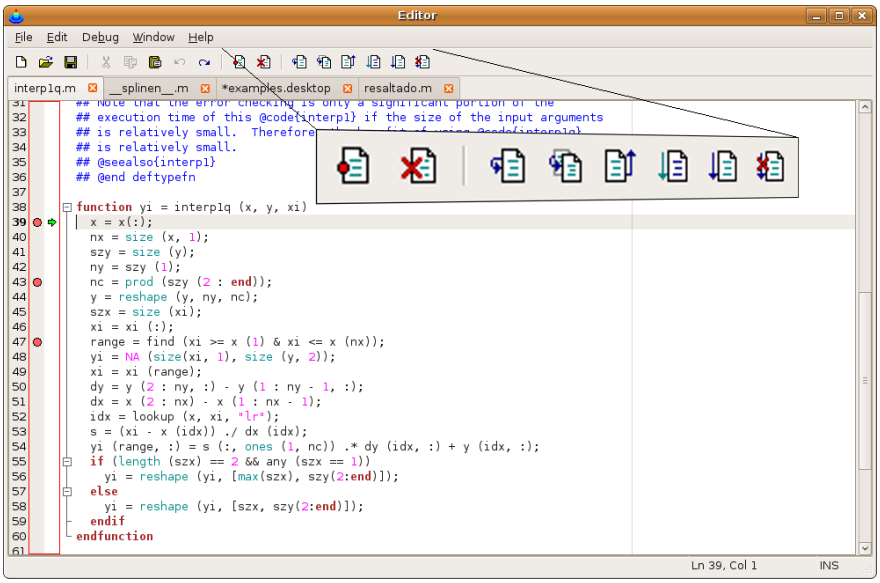
Anexo 8: Opciones de edición y vista exploratoria por pestañas



Anexo 9: Resaltado, completado y plegado de código



Anexo 10: Opciones de depuración y puntos de ruptura



Glosario de Términos

A

Artefacto: Producto tangible resultante del proceso de desarrollo de software. Algunos artefactos como los casos de uso, diagrama de clases u otros modelos UML ayudan a la descripción del diseño del software.

Asistentes matemáticos: Programas para computador diseñados con intencionalidad pedagógica, que permiten el trabajo con: el cálculo numérico y simbólico, la dinamización de la geometría, la gestión de datos, el análisis gráfico de funciones, etc.

B

Binding: Es una adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita.

C

Callback: Puntero a una función, con este mecanismo si se quiere procesar una determinada función cada vez que ocurre un evento en un objeto, lo que se hace es pasar un puntero de otra función a la función deseada y será esta la que se encargue de llamar al callback en el momento apropiado.

Ciclo de desarrollo: Es un proceso por el cual los analistas de sistemas, los ingenieros de

software, los programadores y los usuarios finales elaboran sistemas de información y aplicaciones informáticas.

Código Fuente: Programa tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de maquina que sí pueda ser ejecutado por el hardware de la computadora.

I

Interfaz Gráfica: En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

Iteración: Ciclo corto de construcción repetitivo.

L

Librería: Término informático para referirse a las bibliotecas de vínculos dinámicos.

P

Plegado de Código: Característica de algunos editores de código fuente y los IDE que permite al usuario de forma selectiva ocultar y mostrar las secciones de un archivo.

Plugins: Aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la Interfaz de Programación de Aplicaciones(API).