**PRÁCTICA GENERAL**

**AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE**

**GRADO EN INGENIERÍA INFORMÁTICA**

**GRUPO - M**

**Jesús Garcerán Sáez**

**Sergio Martín Vaquero**

**Daniel Aguado Gala**

**Alberto Hernández Jiménez**

**…**

**ÍNDICE**

1. **Introducción**
2. **Parte I. Mantenimiento del Software**
   1. Requisitos
   2. Planificación de tareas
      1. Identificar tareas
      2. Asignar tareas a personas
      3. Definir un diagrama de PERT
      4. Calendario real
3. **Parte II. Gestión de la configuración**
   1. Herramienta de control de versiones
   2. Conflictos de versiones
4. **Parte III. Pruebas del Software**
   1. Técnica de caja negra utilizada
   2. Procedimiento seguido para la identificación
   3. Casos de prueba identificados
5. **Conclusión**
6. **INTRODUCCIÓN**

Esta práctica se basa en la realización de una calculadora científica guiándonos por la planificación de proyectos dada en clase. Consideramos la calculadora un proyecto puesto que es un conjunto de etapas, actividades y tareas para alcanzar un objetivo, que implica un trabajo no inmediato en un plazo relativamente largo.

Lo primero que hicimos en este proyecto fue la planificación de la práctica. Definimos el tipo de equipo y los requisitos. Tras esto, comenzamos a realizar los diagramas de casos de uso y HTA y definimos las tareas a realizar. Realizamos una planificación de tareas con su correspondiente asignación y completamos el diagrama PERT.

Tras hacer el diagrama PERT, comenzamos la realización de la implementación. A la vez, íbamos actualizando el calendario cada día que trabajábamos en el proyecto con el fin de ver si cumplíamos la planificación. La implementación la hicimos usando GitHub como herramientas de control de versiones.

Por último, al terminar la implementación, definimos los casos de prueba basándonos en la técnica de caja negra de “Partición de clases de equivalencia”. Realizamos los casos de prueba y comprobamos los resultados obtenidos con los esperados. Tras esto, completamos la documentación e hicimos la presentación para clase.

**TIPO DE EQUIPO**

Para la realización de esta práctica hemos contado con un tipo de equipo de estilo **Descentralizado Democrático (DD).** Este organigrama de equipo tiene las siguientes características:

* No tiene líder de grupo permanente, en función de la tarea manda uno u otro.
* Todas las decisiones sobre la práctica se toman en grupo por consenso.
* La comunicación en el grupo se hace de manera horizontal, y las opiniones e ideas de todos los integrantes del grupo tienen el mismo valor.

Con este tipo de quipo hemos conseguido tener una moral más alta en el grupo y una mayor satisfacción. La comunicación ha sido constante.

Por tanto, el paradigma de organización ha sido un **paradigma abierto**, con las siguientes características:

* Mucha comunicación en el grupo.
* Se cuenta con el grupo para la toma de cualquier decisión.

Hemos intentado funcionar con un **Desarrollo Ágil**, puesto que somos un equipo pequeño. De esta manera, la práctica ha ido avanzando poco a poco sin tener problemas de testing o de ofrecer al cliente un programa que no se ajusta a lo pedido.

Los equipos ágiles se caracterizan por:

* Una gran autonomía al a hora de trabajar por parte de todos los integrantes.
* Mínima planificación de las tareas, vamos avanzando poco a poco.
* Realización de breves reuniones diarias en equipo para coordinar el trabajo diario.
* Se considera clave la auto-organización continua y la colaboración entre todos los integrantes del grupo.
* Obtenemos nuestro enfoque según los requisitos del producto y los estándares.

Junto con el desarrollo ágil realizamos “reuniones diarias” para ver como íbamos avanzando en el proyecto, basándonos en la guía SCRUM.

1. **MANTENIMIENTO DEL SOFTWARE**

La parte inicial de la práctica. En este apartado, buscamos obtener la información necesaria para poder comenzar a realizar el proyecto. Por tanto, realizamos una planificación del proyecto, identificando las partes más importantes para comenzar con buen pie:

* Requisitos. Son los servicios que tendrá que proporcionar el sistema y de sus restricciones operativas. Reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema.
* Planificación de tareas. Dividimos el proyecto en tareas, asignando a cada participante del grupo una serie de estas. Además, realizamos una planificación de cuanto tiempo le dedicaremos a la tarea con el fin de poder entregar el proyecto a tiempo. Por último, comparamos la planificación realizada al principio con el calendario real del proyecto, para ver las diferencias y errores que hemos tenido en cuanto a planificación.

1. **Requisitos**

El concepto de requisito es bastante ambiguo. Se puede definir como “lo que debe hacer el sistema”, aunque depende mucho de lo que entendemos el equipo de desarrollo. Dentro de los requisitos definimos dos tipos de requisitos, los funcionales y no funcionales.

REQUISITOS FUNCIONALES

Son declaraciones de los servicios que debe proporcionar el sistema, de la manera que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares.

Mediante consenso, hemos definido los siguientes requisitos funcionales, ya que son los que entendimos a partir de la información dada por el cliente.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requisito | Tipo | |
| RF01 | Paréntesis (jerarquía de operaciones) | | Jerarquía |
| RF02 | Impedir la utilización de dos signos seguidos (formulas erróneas) | | Jerarquía |
| RF03 | Tener la funcionalidad completa por teclado (por ejemplo, para realizar el inverso pulsar “i”). | | General |
| RF04 | Permitir realizar operaciones con números decimales. | | General |
| RF05 | Poder borrar un carácter o una operación | | General |
| RF06 | Poder realizar operaciones científicas (inverso, raíz cuadrada, etc.) | | General |
| RF07 | Introducir números por teclado | | General |
| RF08 | Realizar operaciones básicas de calculadora (suma, resta, multiplicación y división). | | Jerarquía |

REQUISITOS NO FUNCIONALES

Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requisitos no funcionales a menudo se aplican al sistema en su totalidad.

Al igual que los requisitos funcionales, los hemos definid o de manera grupal a partir de la información precisada por el cliente.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requisito | Tipo | |
| RNF01 | El lenguaje de programación será Java | | General |
| RNF02 | Colocación habitual de botones numéricos en la calculadora | | Interfaz |
| RNF03 | Añadir interfaz con apariencia de calculadora | | Interfaz |
| RNF04 | Debe ser una aplicación robusta y tratar errores | | General |
| RNF05 | Visualizar la operación que estas realizando en la parte superior de la pantalla de la calculadora. | | Interfaz |
| RNF06 | Poder redimensionar la calculadora con un diseño “responsive” | | General |
| RNF07 | Ofrecer botón de ayuda al usuario | | Interfaz |

REPRESENTACIÓN DE REQUISITOS

Una vez enumerados los requisitos candidatos, necesitamos, para entenderlos mejor, comprender el Contexto del Sistema y capturar los Requisitos Funcionales. Esto es algo que se suele hacer al comenzar un programa, y sirve para clarificar las ideas y compartirlas con el cliente.

* **Comprender el Contexto del Sistema (Diagrama HTA)**

Para comprender el contexto del sistema (el entorno en el que se enmarcará el sistema), hemos realizado un diagrama HTA.



* **Capturar los requisitos funcionales (Diagrama de casos de uso)**

El diagrama de casos de uso permite a los desarrolladores de software y a los clientes llegar a un acuerdo sobre los requisitos. Cada caso de uso representa la forma en que los actores usan el sistema.

El diagrama realizado ha sido el siguiente:



Cada caso de uso ha sido descrito de la siguiente manera:

|  |
| --- |
| **Caso de uso: “Realizar operación básica”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará un número (con una serie de pulsaciones de botón indeterminado de pulsaciones de botón) y tras esto seleccionará la operación deseada a realizar (suma, resta, multiplicación, división, etc.). Por último, pulsará otra serie de botones para obtener un número, o, por el contrario, empezaremos a usar jerarquía (con paréntesis). |

|  |
| --- |
| **Caso de uso: “Realizar operación científica”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará un número (con una serie de pulsaciones de botón indeterminado de pulsaciones de botón) y tras esto seleccionará la operación deseada a realizar (raíz cuadrada, inverso, doble, etc.). La operación se hará directamente y el sistema mostrará el resultado de la operación. |

|  |
| --- |
| **Caso de uso: “Pulsar botón de ayuda”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará el botón ayuda y se mostrará en pantalla una nueva pestaña en la que se le explicará al usuario como usar la calculadora, las operaciones que puedes realizar con ella y todo tipo de información relevante sobre el sistema. |

|  |
| --- |
| **Caso de uso: “Borrar carácter”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará el botón “CE” y el sistema borrará el último carácter que ha escrito el usuario. Se puede pulsar el botón tantas veces como sea necesario siempre que existan caracteres. |

|  |
| --- |
| **Caso de uso: “Borrar operación”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará el botón “C” y el sistema borrará la operación realizada hasta el momento, es decir, reseteará la operación realizada hasta el momento para empezar de cero. El sistema borrará toda la información almacenada hasta el momento y permitirá al usuario escribir otra operación. |

|  |
| --- |
| **Caso de uso: “Pulsar botón igual”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará el botón igual y el sistema realizará la operación o la suma de operaciones con su correspondiente jerarquía. El sistema, una vez realizada la operación, mostrará el resultado en pantalla. |

|  |
| --- |
| **Caso de uso: “Usar jerarquía (paréntesis)”** |
| Actor: Usuario |
| Descripción:  El usuario pulsará el paréntesis izquierdo, y tras esto realizaremos una operación (ya sea compleja o básica) en la que podremos escribir otro paréntesis en su interior. Una vez escrita la operación, pulsaremos el botón paréntesis derecho para obtener la jerarquía y poder obtener el resultado. |

1. **Planificación de tareas**

El objetivo del proyecto es el de realizar una calculadora científica, cuya funcionalidad principal sea la de la jerarquía de operaciones. Debe ser una calculadora para ordenador, por tanto, debe tener la funcionalidad básica mediante botones, aunque también podría usarse el teclado. Obviamente, la calculadora debe realizar todas las operaciones básicas que realiza cualquier calculadora, y permitir trabajar con todo tipo de números.

1. **Identificar tareas**

Lo primero que realizamos es la identificación de tareas a realizar por el equipo. Este es uno de los primeros trabajos que se hizo para la correcta planificación del proyecto. Las tareas a realizar se definieron de manera conjunta por todos los integrantes del grupo y con un nivel de detalle medio. Las tareas que definimos fueron las siguientes:

1. Definir los requisitos de usuario.
2. Definir tipo de equipo.
3. Definir las tareas a realizar.
4. Organizar las tareas entre los integrantes del equipo.
5. Hacer el diagrama de casos de uso.
6. Hacer el diagrama HTA.
7. Hacer el prototipo en papel de la interfaz gráfica.
8. Hacer el prototipo “detallado”.
9. Hacer el diagrama PERT.
10. Planificación de la realización de pruebas.
11. Realizar la interfaz de la calculadora.
12. Implementar la funcionalidad de los botones numéricos (cuando pulsas que salga en pantalla).
13. Implementar la realización de operaciones básicas (suma, resta, multiplicación y división).
14. Implementar el botón de igual para la realización de la operación.
15. Implementar “no permitir más de un carácter de operación a la vez”, permitir solo realizar una operación.
16. Implementar el borrado de un solo carácter y de una operación.
17. Implementar la realización de las operaciones científicas (inverso, doble, raíz cuadrada).
18. Implementar la funcionalidad de botones numéricos por teclado.
19. Implementar la funcionalidad avanzada por teclado.
20. Mostrar en la parte superior de la pantalla la operación que estamos realizando.
21. Crear la pestaña ayuda para el usuario, informando de los “atajos de teclado” y de como funciona la calculadora.
22. Implementar la realización de priorización de operaciones mediante paréntesis.
23. Realizar la documentación del proyecto.
24. Casos de prueba y resultados obtenidos.
25. **Asignar tareas y estimar**

Una vez definimos las tareas, realizamos una asignación de tareas a los distintos integrantes del equipo. Esta asignación se realizo, al igual que la definición de tareas, de manera conjunta y cada persona eligió la parte que iba a realizar, de manera que nadie asigno a otra persona una tarea.

Esta asignación se hizo siguiendo la filosofía de equipo de **Descentralizado Democrático**, como hemos comentado anteriormente.

Una vez asignadas las tareas, pasamos a la estimación de cuanto tiempo pensábamos que teníamos que dedicarle a cada tarea de las definidas. De tal forma, se realizo una planificación temporal. Definimos la siguiente tabla:



Esta estimación se hizo en base a nuestro conocimiento y experiencia como desarrolladores. Además, se tuvo en cuenta el tiempo que se le iba a dedicar (aproximadamente) al trabajo, ya que, al no poder dedicarle totalmente nuestro tiempo diario a este proyecto, le íbamos a dedicar intervalos de tiempo variables.

1. **Definir diagrama PERT**

Son diagramas basados en redes de precedencia. Es una representación grafica del proyecto que relaciona las actividades, y permite visualizar las que son críticas. El diagrama PERT definido, según las tareas y nuestra estimación temporal, fue el siguiente:

**DIAGRAMA PERT**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Actividades | Precedentes | Duración | TEi | TLi | TEj | TLj | HT | HL | HI | RRHH |
| A | - | 2 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | Todos |
| B | A | 1 | 2 | 2 | 3 | 3 | 0 | 0 | 0 | Todos |
| C | B | 1 | 3 | 3 | 4 | 4 | 0 | 0 | 0 | Jesús |
| D | C | 1 | 4 | 4 | 5 | 5 | 0 | 0 | 0 | Todos |
| E | C | 2 | 4 | 4 | 6 | 6 | 0 | 0 | 0 | Sergio, Jesús |
| F | C | 1 | 4 | 4 | 8 | 8 | 3 | 3 | 3 | Jesús |
| G | A | 1 | 2 | 2 | 8 | 8 | 5 | 5 | 5 | Alberto dlP |
| H | D | 1 | 5 | 5 | 6 | 6 | 0 | 0 | 0 | Alberto |
| I | H | 2 | 6 | 6 | 8 | 8 | 0 | 0 | 0 | Alberto |
| J | E | 2 | 6 | 6 | 8 | 8 | 0 | 0 | 0 | Alberto |
| K | J,I,F,G | 3 | 8 | 8 | 11 | 11 | 0 | 0 | 0 | Sergio |
| L | K | 2 | 11 | 11 | 13 | 19 | 6 | 0 | 0 | Alberto |
| M | N | 2 | 13 | 13 | 15 | 15 | 0 | 0 | 0 | Alberto |
| N | K | 2 | 11 | 11 | 13 | 13 | 0 | 0 | 0 | Alberto |
| O | M | 3 | 15 | 15 | 22 | 22 | 4 | 4 | 4 | Alberto |
| P | L | 2 | 13 | 20 | 22 | 22 | 7 | 7 | 0 | Alberto |
| Q | M | 3 | 15 | 15 | 18 | 18 | 0 | 0 | 0 | Alberto |
| R | K | 2 | 11 | 11 | 13 | 18 | 5 | 0 | 0 | Alberto |
| S | R | 3 | 13 | 18 | 22 | 22 | 6 | 6 | 0 | Alberto |
| T | Q | 1 | 17 | 17 | 18 | 18 | 0 | 0 | 0 | Alberto |
| U | K | 1 | 11 | 11 | 22 | 22 | 10 | 10 | 10 | Alberto |
| V | T | 3 | 19 | 19 | 22 | 22 | 0 | 0 | 0 | Alberto dlP |
| W | U,P,S,O,V | 3 | 22 | 22 | 25 | 25 | 0 | 0 | 0 | Jesús |
| X | W | 2 | 25 | 25 | 27 | 27 | 0 | 0 | 0 | Todos |

El diagrama PERT se quedo de la siguiente manera:



Las conclusiones obtenidas fueron que las tareas L y R eran las únicas que tenían un poco de margen. El resto eran tareas críticas que se deberían realizar según el tiempo estimado y no deberíamos sobrepasarlo.

1. **Calendario real**

Una vez realizada la planificación comenzamos a realizar el proyecto, en cuanto a desarrollo se refiere. Cada día que se trabajaba en el proyecto era apuntado en un calendario con el fin de ver si se cumplía la estimación realizada.

**CALENDARIO**

Marzo



Abril



Los problemas surgidos fueron los siguientes:

1. **GESTIÓN DE CONFIGURACIÓN**

Le gestión de configuración es el conjunto de actividades de seguimiento desde el inicio hasta el final del proyecto software.

1. **Herramienta de control de versiones**

La herramienta que hemos usado para el control de versiones ha sido GitHub. GitHub es una plataforma de desarrollo colaborativo de software, que sirve para alojar proyectos, utilizando el sistema de control de versiones Git.

Está herramienta nos proporciona una serie de herramientas bastante útiles para el trabajo en equipo. Destacamos entre ellas:

* Una wiki para el mantenimiento de las distintas versiones.
* Un sistema de seguimiento de problemas para detallas problemas o sugerencias con respecto al proyecto entre el equipo.
* Una herramienta de revisión de código que permite añadir anotaciones o debatir en cualquier punto de un fichero.
* Un visor de ramas donde se pueden comparar los progresos realizados.

**Estrategia de versionado**

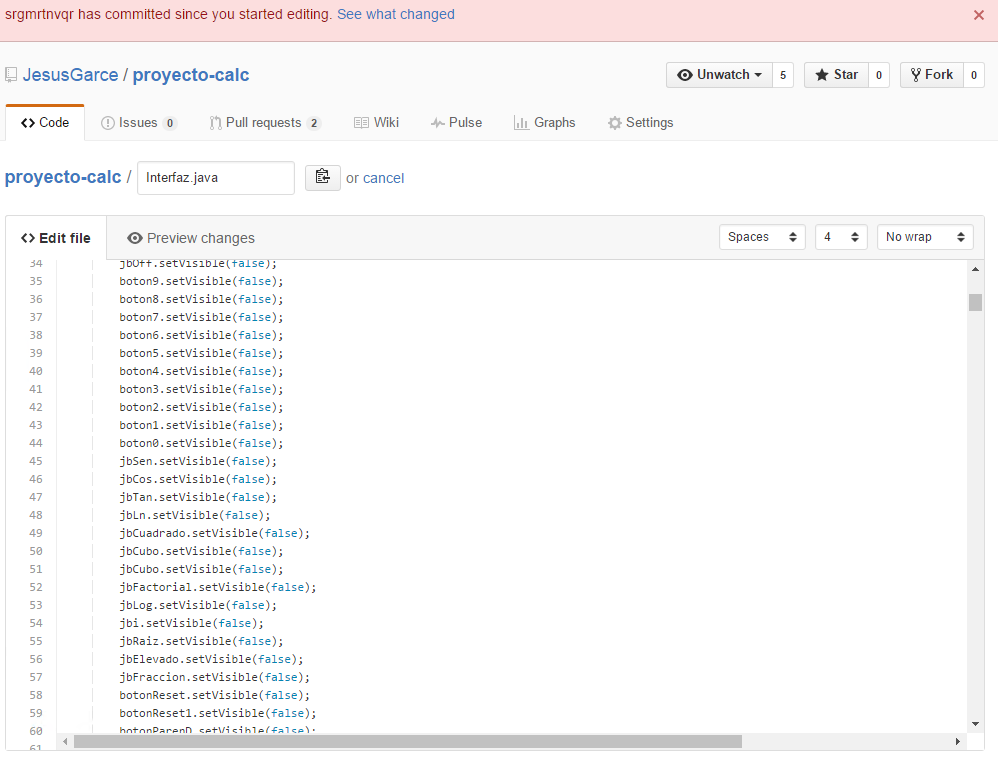
Esta herramienta sigue una estrategia de versionado **Copiar-Modificar-Mezclar**. En esta estrategia, en el caso de un conflicto por que un usuario quiere subir un archivo desincronizado debe sincronizarlo para poder subirlo. De está forma se solucionan los problemas de versiones entre distintos usuarios.

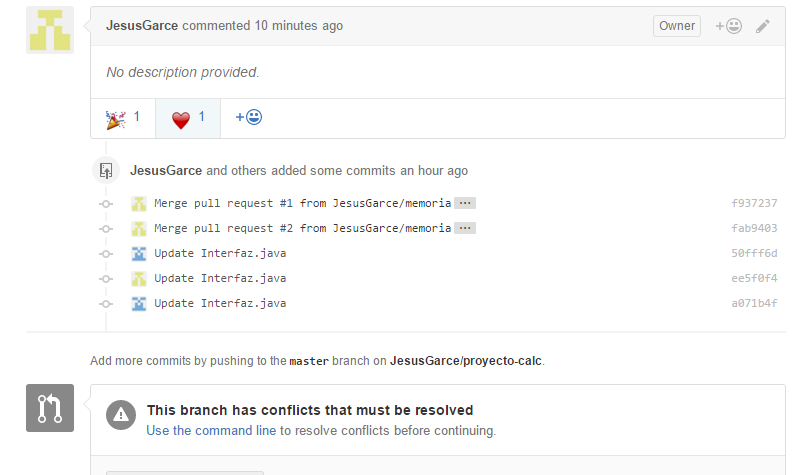
**Tipo de repositorio**

Es un sistema de control de versiones **distribuido**. Esto es, el repositorio está distribuido por todos los participantes, y estos tienen en local toda la información que necesitan. Sin embargo, no ocupa mucho espacio, ya que tendremos únicamente las partes que nos interesa (la rama de la que hemos partido).

Este tipo de repositorio provoca que ramas diverjan. Es un problema que se soluciona haciendo “merge”, uniendo los cambios sobre una línea principal.

1. **Conflictos de versiones**





1. **PRUEBAS DEL SOFTWARE**

Las pruebas del software son importantes con el fin de solucionar fallos. Un fallo tiene muchísimos costes en software.

Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, y tiene una serie de características:

* Es un proceso de ejecución de un programa con el fin de descubrir un error.
* Un buen caso de prueba tiene una alta probabilidad de encontrar un error
* Una prueba tiene éxito si descubre un error no detectado con anterioridad.

Las pruebas del software tienen una serie de principios:

1. A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos
2. Deberán planificarse muchos antes de que empiecen.
3. El principio de Pareto es aplicable (80% resultado, 20% esfuerzo).
4. Se deben incluir tanto entradas correctas como incorrectas.
5. Las pruebas deberían empezar por lo pequeño y progresar a lo grande.
6. Imposibilidad de hacer pruebas exhaustivas.
7. Realización de pruebas por equipos independientes.
8. **Técnica de caja negra**

El enfoque de caja negra tiene una serie de principios:

* Se centra en los requisitos funcionales del software.
* Permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa.
* No es una alternativa a las técnicas de prueba de caja blanca.
* Se rata de un enfoque complementario que intenta descubrir diferentes tipos de errores.
* Tiende a aplicarse durante fases posteriores a la prueba.

Detecta errores en las siguientes categorías: funciones incorrectas, errores de interfaz, errores en estructuras de datos o accesos a bases de datos, errores de rendimiento y errores de inicialización.

La técnica que vamos a usar es la de **Partición en clases de equivalencia**. Es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Una **clase de equivalencia** representa un conjunto de estados válidos o no válidos para condiciones de entrada.

Como, en nuestro caso, una condición de entrada requiere un valor específico, definimos una clase de equivalencia válida y una no válida, y en nuestra clase de equivalencia válida solo tendremos un resultado.

Hemos elegido esta técnica puesto que, al ser una calculadora lo que vamos a tratar, es más difícil descubrir cuales son los “límites” de las clases, ya que en una calculadora un resultado debe ser el correcto, y cualquier otro resultado es erróneo. Por tanto, vemos más adecuado usar clases de equivalencia.

1. **Identificación de casos de prueba**

Para identificar los casos de prueba vamos a identificar cuales son las clases que tenemos, con el fin de tratar todos los casos que podemos identificar en una calculadora científica. Separamos en diversos bloques:

* **Operaciones básicas.** Una calculadora, al escribir nuestra operación básica (suma, resta, multiplicación, división) debe darnos el resultado correcto.
* **Jerarquía de operaciones.** 
  + Al escribir distintas operaciones en una misma línea se debe aplicar la jerarquía de operaciones
  + Al escribir paréntesis para marcar que operación se debe hacer antes y después debe tener un resultado correcto.
* **Operaciones científicas.**
  + Al realizar operaciones científicas con dos operandos debe dar un resultado correcto.
  + Al realizar operaciones científicas con un operando el resultado debe ser el adecuado.
* **Operaciones científicas + Jerarquía de operaciones.**
  + Al escribir operaciones científicas junto con operaciones básicas debe dar el resultado, tratando ambas operaciones.
  + Al escribir diversas operaciones científicas deben ser tratadas adecuadamente para dar el resultado correcto.
  + Al escribir operaciones científicas junto con operaciones básicas y paréntesis el resultado debe ser el adecuado.
  + Al escribir operaciones científicas junto con paréntesis el resultado debe ser correcto.
* **Funcionalidad extra**. Se debe comprobar que la funcionalidad de la calculadora, aparte de la ejecución de operaciones, actúa de manera correcta (hacer borrado de un carácter, de una expresión, permitir poner puntos, etc.)
* **Errores.** Se deben tratar los distintos errores que pueda tener el usuario al escribir la expresión a realizar por la calculadora.
* **Funcionalidad teclado.** Se debe comprobar que funcionan todas las teclas de la calculadora pulsando con el teclado.
* **Interfaz.** La interfaz debe tener una funcionalidad correcta sin errores visibles por el usuario. Todos los botones deben funcionar de manera adecuada.

1. **Casos de prueba identificados**