**Documento de Diseño Proyecto 3 Entrega 1**

**Iteración final de diseño Actualizada proyecto 3:**

1. **Roles:**
2. Proyecto<<estructurer>>
3. Participante<<information holder>>
4. Actividad<<structurer
5. Registro<<info holder>>
6. Reporte<<structurer>
7. vi. Repo git/service
8. Archivo/info holder
9. Almacenamiento/actividades
10. Fecha/info
11. Cronometro/ service
12. Agregador participante/ strcuturer
13. Tiempo/ info holder
14. Manejadoractividades/structurer
15. Mnaejadoregistro/strcuturer
16. Manejadorparticpantes/strcutrer
17. Manejador version/ strcuturer
18. Registraractividad/ info holder
19. TIpoactividad/ info holder iv. Manejadorarchivos/ strcuturer
20. Participante inicial/ info holder
21. Manejador WBS <<Strcuturer>>
22. Agregador WBS <<Infro Holder>>
23. GraficaBarras <<Info Holder>>
24. AgregadorActividadWBS<<Strcuturer>>
25. ConexiónTareasWBS<<Strcuturer>>
26. RporteWBS<<Structurer>>
27. **Responsabilidades**

|  |  |
| --- | --- |
| Responsabilidad | Rol |
| Crear proyecto | proyecto |
| Agregar particpante | particpante |
| Registrar actividad | participante |
| Mostrar reporte | reporte |
| Tomar tiempo | actividad |
| Modificar datos | participante |
| Mirar reporte | participante |
| Manejar versiones | Repo git |
| Escribir actividad | archivo |
| Almacenar actividad | almacenamiento actividades |
| Registrar fecha inicio | fecha |
| Registrar fecha final | fecha |
| Pausar tiempo | cronometro |
| Agregar particpante | agregador psrticpante |
| Tomar tiempo activdad | tiempo |
| Verificar info actividades | manejador actvidades |
| Verficiar registros | manejador registro |
| Mirar particpantes | manejador particpantes |
| Verificar versiones | manejador versiones |
| Registrar nombre | registrar actividad |
| Definie tipo | tipoactividad |
| Veficiar archivo plano | manejadores archivos |
| Crear proyecto | particpante inicial |
| Organizar proyectos y tareas internas | Manejador WBS |
| Definir nombre y descripción WBC | Manejador WBS |
| Definir tiempo aprox actividad | actividad |
| Indicar actividad WBS | AgregadorActividadWBS |
| Generar graficas | GraficaBarras |
| Suma total tiempo | Manejador WBS |
| Avance proyecto | reporte |
| planeación | Manejador WBS |
| Resumen Proyecto | Proyecto |
| Reporte equipo | Proyecto |

**UML de iteracion en el repositorio por separado**

**Decisiones de Diseño Proyecto 3:**

1. **Patrón**
   1. Con respecto, al tema de patrones, se tiene que decir que se planea usar alguno de los patrones mencionados durante la clase. Antes de mencionar cuales son los patrones que se planean utilizar, queremos aclarar cual va a ser su función en el proyecto. Es decir, cual es la necesidad que creemos existe para poder usar estos patrones. Esta función realmente es básica. La función que estos patrones van a tener en nuestro proyecto es poder optimizarlo. Esto hace referencia a poder realizar ciertas funciones, que mas adelante se van a tratar, que permiten que el proyecto sea optimo. Ahora bien, después de haber dejado clara la función de estos patrones, vamos a hablar de cuáles son los que vamos a tratar de usar en la implementación. El primer patrón seleccionado fue el patrón iterador. Este patrón básicamente ayuda a recorrer estructuras. La selección de este patrón se debe, a como se dijo anteriormente, para poder recorrer ciertas estructuras. Esto nos va a ayudar a realizar recorridos sobre los proyectos o participantes. Como se dijo antes, se busca que la implementación sea lo mas optimo posible. Por esta razón, es que implantamos el iterador. El segundo patrón que pensamos en implementar es el patrón Mediator. Este patrón, conecta clases. La elección de este patrón se debe a que necesariamente tenemos que conectar clases de nuestro proyecto de cierta manera. Por ejemplo, cuando permiso un proyecto, tenemos que traerlo del archivo leerlo y mandarlo a la clase proyecto para que se transforme. En este caso, ya estaríamos usando el patrón mediator. Esto nos va a ayudar a la implementación, sea optima, y además se logra tener un mejor manejo de datos a lo largo del trabajo. Esto con la intención de tener la mejor conexión posible. Y como ultimo patrón seleccionado fue el patrón Observer. Este patrón básicamente avisa de cambios de estado de la clase en la que se encuentre. En este caso, este patrón ya está implementado de cierta manera. Esto se debe a que en la implementación ya se encuentra el patrón. Puesto que, para hacer funcionar los botones de la interfaz gráfica, se debía hacer que un objeto reconociera la relación del usuario con la pantalla. Por lo que se podría decir que este patrón ya está usándose a lo largo del proyecto.
2. **Errores**
   1. Con respecto al tema de manejo de errores, se tiene que hablar de como se espera que se comporte el programa cuando suceda algún error inesperado. Antes de hablar de las estrategias seleccionadas, es necesario hablar de cuál es la función de estas estrategias en nuestro programa. La función de estas estrategias como su nombre lo indica es la prevención de errores. Esto con la intención de no sucedan fallas. La prevención de esto errores resulta necesario para que no haya problemas con la ejecución cuando uno de esto ocurra. Por esta razón, es que preferimos manejarlos con antelación para que no resulte en mayores problemas. Ahora bien, luego de haber tratado por encima cual es la importancia del manejo de errores par a nosotros se tiene que hablar de cuales fueron los que se seleccionaron. La primera estrategia que planeamos implementar es la estrategia de prevención de errores. Esta estrategia, hace referencia, como su nombre lo indica, a prever errores que pueden llegar a suceder en el programa. Esta estrategia la planeamos utilizar principalmente cuando el usuario está en contacto con el programa. Esto hace referencia a que cuando se requiera la mano del usuario se tenga una prevención de errores. Algunos de estas se planean modificar y otros alertar al usuario. Un ejemplo de esto puede suceder cuando el usuario este tecleando el nombre del proyecto. Esta estrategia, planeamos hacer ciertas excepciones a lo largo del programa para que se pueda prevenir este error común. Como segunda estrategia a utilizar, elegimos la inacción. En algunos casos par a nuestro programa no será necesaria responder al error. Esto puede suceder cuando el usuario no selecciona proyecto para ver información. Dicho esto, esta estrategia hace los que su nombre indica, nada ante un error. Por último, la estrategia seleccionada fue Balk. Esta estrategia realiza un aviso al usuario cuando algo salió mal. Realmente esto y sol que buscamos en nuestro programa en algunos casos que el usuario se de cuenta del error que no podemos solucionar. Esto se debe a que normalmente este tipo de errores fueron culpa del usuario. Dicho esto, estas fueron la estrategia que se planean utilizar en el proyecto.
3. **Principios de Diseño**
   1. Con respecto al tema de principio de diseño, se habla de cómo está estructurado el programa. Esto hace referencia a como están conectadas las clases entre sí y dentro de sí mismas. Antes de hablar de cual principio fue el que seleccionamos uy tratamos de implementar, surge la necesidad de hablar del porque es importante para nosotros. En este caso, no es tan complicado. Puesto que, ambos tienen una cierta funcionalidad o posibilidad de uso. Esta funcionalidad es buscar el bajo acoplamiento y la alta cohesión. Ahora bien, esta funcionalidad porque es importante en nuestro proyecto. Esto no sirve para poder realizar actualización a nuestro proyecto. Un ejemplo de esto es esta misma entrega donde toca actualizar el proyecto. Entonces, estos principios nos ayudan de cierta manera que esta actualización no sea tan complicada. Ahora bien, luego de haber hablado de la importancia de estos principios en nuestro proyecto vamos a decir cuál fue la elección. En este caso, la elección o el principio que seleccionamos fue el principio Solid. Donde en cada uno de sus campos vamos a tratar de implementarlo de mejor manera. En el caso de Single responsiblity, se intentará que cada clase haga los suyo. Por ejemplo, en la clase proyectos que solo se hagan cosas de proyectos. Esto ya está implantado, peor se espera mantener esto después de las actualizaciones. Luego, el punto más importante de open closed principles. Vamos a tratar de que sea el que guie la implementación. Puesto que, es el que permite actualizar el proyecto de mejor manera. El punto de Liskov, no se espera tenerlo tan calor. Sin embargo, se espera que si se utilizar herencia se usara en el proyecto. Luego, para el caso de interfaz segregación. Se espera que se maneje la herencia de cierta manera que se maneta el principio. Y por ultimo la dependencia de inversión, resulta fundamental. Par manejar el proyecto para que se pueda aplicar a varios contextos dentro del mismo. Es decir, que se maneje el acoplamiento de la mejor manera.