Juan Alberto Gutiérrez Canto

Análisis de los datos hasta el momento, sobre el desarrollo del software. En un entorno del proceso PSP.

R4

Reporte

**Datos Generales**

|  |  |
| --- | --- |
| **Materia** | Proceso personal de software |
| **Tarea** | Reporte R4 |
| **Fecha de Liberación** | 18/04/2016 |
| **Fecha de Entrega** | 02/05/2016 |
| **Descripción General** | Entender la densidad y el tipo de defectos introducidos y encontrados durante el desarrollo de programas.  Analizar el tiempo y el tamaño estimado y real de los programas para identificar las mejoras obtenida y las áreas de oportunidad. |
| **Actividad** | Analizar e interpretar las gráficas generadas por la herramienta de PSP para poder así entender que es lo que se hizo y por qué se está generando estos resultados.  Obtener las gráficas de estimación tiempo y el tamaño estimado y el real de los programas, analizarlas e interpretarlas. Además de escribir un listado de los PIP´s, las mejoras obtenida y las áreas de oportunidad. |
| **Catedrático** | Patricia Eréndira Benavides Muratalla. |

**Alumno**

|  |  |  |
| --- | --- | --- |
| **No. Matricula** | **Apellidos** | **Nombre(s)** |
| 24400063 | Gutiérrez Canto | Juan Alberto |

**INDICE**

Tabla de contenido

[OBJETIVOS 3](#_Toc450264677)

[PROCESO DE ANALISIS 3](#_Toc450264678)

[ANALISIS DE DEFECTOS 4](#_Toc450264679)

[TOTAL, DE DEFECTOS POR PROGRAMA 4](#_Toc450264680)

[DEFECTOS INYECTADOS EN DISEÑO POR PROGRAMA 6](#_Toc450264681)

[DEFECTOS INYECTADOS EN CODIFICACION POR PROGRAMA 7](#_Toc450264682)

[DEFECTOS REMOVIDOS EN COMPILACION POR PROGRAMA 8](#_Toc450264683)

[DEFECTOS REMOVIDOS EN PRUEBAS POR PROGRAMA 10](#_Toc450264684)

[DEFECTOS INYECTADOS A LA FECHA POR FASE 11](#_Toc450264685)

[DEFECTOS REMOVIDOS A LA FECHA POR FASE 12](#_Toc450264686)

[ANALISIS DE DEFECTOS MEDIANTE PARETO 13](#_Toc450264687)

[DEFECTOS REMOVIDOS Y FASE DE LOS DEFECTOS 14](#_Toc450264688)

[TIEMPO DE CORRECCION DE DEFECTOS 16](#_Toc450264689)

[TABLAS DE ANALISIS DE DATOS 17](#_Toc450264690)

[DENSIDAD DE LOS DEFECTOS Y FASE DE LOS DEFECTOS 17](#_Toc450264691)

[TIEMPO DE CORRECCION DE DEFECTOS 20](#_Toc450264692)

[ANALISIS DE ESTIMACIONES 21](#_Toc450264693)

[ESTIMACIONES DE TAMAÑO 21](#_Toc450264694)

[ESTIMACIONES DE TIEMPO 23](#_Toc450264695)

[TABLA DE PRODUCTIVIDAD POR PROGRAMA 29](#_Toc450264696)

[DESEMPEÑO PERSONAL AL USAR PSP 30](#_Toc450264697)

[RECOPILACION DE LOS PIP´S A LO LARGO DEL PROCESO 31](#_Toc450264698)

[AREAS DE MAYOR PRIORIDAD PARA HACER MEJORAS EN LA SEGUNDA PARTE DEL CURSO 32](#_Toc450264699)

[CONCLUSIÓN 34](#_Toc450264700)

# OBJETIVOS

Para la mitad del PSP y finales de este curso, el proceso de PSP nos debe de presentar una manera fácil de ver la recopilación, análisis y entender las métricas referentes a la planeación de tamaño, a la planeación de tiempo y productividad para encontrar las mejoras que se han logrado hasta el momento e identificar las áreas de oportunidad para el futuro, siempre pensando en que el proceso se mejora constantemente con los siguientes puntos:

* Comprender los principios de la medición y análisis de procesos.
* Ganar experiencia en la definición de un proceso para uso propio
* Aprender a analizar los datos de un proceso.
* Comprender metas medibles y definir las acciones de mejora correspondientes.

# PROCESO DE ANALISIS

# ANALISIS DE DEFECTOS

Para que analizar la cantidad, el tipo y la fase, así como que tanto tiempo se repite los defectos, bueno pues esto tiene una finalidad elemental, que es la auto mejora del PSP.

Analizaremos los tipos de defectos comprendidos por:

* Total, de defectos por programa
* Defectos inyectados e diseño por programa
* Defectos inyectados en codificación por programa
* Defectos removidos en compilación por programa
* Defectos removidos en prueba por programa
* Defectos inyectados a la fecha por fase
* Defectos removidos a la fecha por fase.

A continuación, se presenta el análisis.

## TOTAL, DE DEFECTOS POR PROGRAMA

Lo primero que analizamos es el total de defectos generados por programa, este se representa en Defects/KLOC por programa.

A lo largo del curso, se presentaron diferentes programas, cada uno con su grado de complejidad, en algunos, mi mal desempeño, hizo que los defectos se dispararan a casi 200 defects/loc’s, pero en la mayoría, se presenta una variante de 10 0 20 loc’s de defectos, analizando la gráfica descubrimos que:

* En el programa 2, es donde tengo la mayor cantidad de defectos por loc’s, esto fue causado por varias cosas. La primera causa es porque, yo alguna vez había hecho un código parecido, y se me hizo fácil empezar a codificar sin hacer nada de diseño, el problema fue que no solamente me di cuenta que el programa era totalmente diferente, si no que las funciones y las librerías que había utilizado en el programa anterior, ya no eran funcionales o no me acordaba de sus nombres. Por lo que, al momento de codificar, solamente invente y trate de adivinar, eso hizo que sobresalieran una gran cantidad de errores.
* Seguido con unos de los promedios más bajas de defectos se presenta el programa 3. Este programa es una variante del proyecto 2, por lo que su realización fue demasiado fácil y solamente tuve que agregar funcionalidades básicas, ya que un proyecto antes (el 2) me había peleado con todas las funciones y diseño del proyecto ya en codificación, para el proyecto 3 fue más fácil solamente añadir unas pocas funcionalidades.
* Desde el proyecto 4, empecé a pensar un poco más en el diseño antes de codificar, lo que al principio si me causo conflictos, en mi tiempo de desarrollo, pero a lo largo de los problemas me facilito la codificación y disminuyo la tasa de defectos por loc’s.
* Para la mayoría de mis defectos, los había encontrado en la parte de codificación, ya que cuando se compilaba o se pasaba a la fase de test, estos eran descubiertos. Después del diseño se empezó a descubrir defectos en la mayoría de las fases, hasta en una ocasión, me di cuenta que me faltaba un requerimiento cuando ya estaba en la fase de pruebas, absurdo pero cierto.

## DEFECTOS INYECTADOS EN DISEÑO POR PROGRAMA

Analicemos los defectos inyectados en la fase de diseño por loc’s.

Para los defectos inyectados en diseño contados en defects/Klos’s por programa, podemos ver la tendencia y descubrir cuándo y por qué falla nuestro diseño, así como comentar cuales fueron las técnicas y procesos que ayudaron a mejorar esta parte del proceso:

* En los programas 1,4,5 no tenemos ningún defecto inyectado en diseño. En el programa 1 no se hizo diseño eso simplifica el por qué no contiene ningún defecto a la falta de diseño, es igual a la falta de defectos o métrica alguna para medirlo.
* Para el proyecto 4,5 si se tuvo un diseño, pero este solamente correspondido a la fabricación de las funciones así, como tomar en cuenta las variables y algunos de los procesos, elementales que se iban a emplear en los proyectos, no se presentó ningún defecto, porque no fue un diseño muy complejo, fue el comienzo de diseñar un proyecto, con las herramientas que ya sabemos utilizar, como son casos de usos y seudocódigos.
* En el programa 2, que fue el más complejo que eh sentido hasta el momento y en el cual me confié demasiado, al hacer el diseño, no tome en cuenta, toda la funcionalidad del proyecto, y lamentablemente esto causo un gran impacto en los defectos. También cabe analizar que mi diseño fue más que básico, y solamente realicé seudocódigo a la ligera, no sabía ni como se llamaban las funciones, ni que valores regresaban, solamente pretendí saberlo, debiéndome asegurar que era lo que realmente hacía, fue un trabajo mal hecho, ya que no obtuve nada bueno de ese diseño y tuve más defectos que los que esperaba.

## DEFECTOS INYECTADOS EN CODIFICACION POR PROGRAMA

Analicemos los defectos en la tercera fase de desarrollo del software, la codificación, estos están representados en la gráfica como defects/KLoc.

Para una de las partes más esenciales que es la codificación, y en donde los autores, mencionan que es la fase donde se cometen la mayor parte de los defectos. Por mi parte podría decir que se presentan por la falta de experiencia y de procesos, ya que muchos de estos, no son más que cosas que se pueden evitar. Por ejemplo:

* Empezando por el programa 2, denoto que el mal diseño, causa que dentro de este programa, se obtuvo 115 defects/Loc’s en codificación, ya que al momento de diseñar mientras de codifica, se inyectan muchos defectos de sintaxis, así como también defectos de funciones, ya que no tenía ni idea de que valores enviar o no a las funciones y que valores me devolvería, otro problema más que puedo mencionar aquí es que al diseñar en la fase de codificación, causo que los casos de pruebas fronteras, no los tuviera en cuanta, no tenía ni la menor idea de que existieran y a la hora de la fase de test, los descubrí. A falta de un buen diseño, estos errores se cargaron a la fase de codificación, todo esto causo una gran carga de defectos en el código del programa 2.
* Parte fundamental de los defectos que inyecto en codificación, son por que no se hace una revisión del código con intención de encontrarlos, esto es, solamente se revisa intentando corregir cualquier error de lógica y diseño, no centrándose en la sintaxis o las funciones.
* Aun que podemos observar que la gráfica presenta una variante del 20 LOC´s por defectos de programa, puedo aclarar que es un rango muy limitado del cual eh podido salir hasta el programa 6, en el cual tuve un diseño más concreto y una planificación más compleja, acerca de lo que iba a hacer.

## DEFECTOS REMOVIDOS EN COMPILACION POR PROGRAMA

Ahora comenzaremos a ver, la parte de defectos removidos, así como las fases en donde se removieron. Empezamos por ver la gráfica de defectos removidos en compilación.

En esta grafica podemos observar los defects/Kloc’s removidos por programa, para el programa 1 siempre presentaremos un valor de 0, ya que se hizo con un proceso no definido, para los demas programas descubrimos que:

* La mayoría de los defectos que se remueven en la fase de compilación son de tipo, sintaxis. Esto es debido a que el programa no corre, ya que el compilador, encuentra falta de coherencia en la sintaxis. De los errores que más eh cometido durante este proceso, son:
  + No poner;
  + Escribir mal una variable
  + Escribir mal una función
  + Poner mayúsculas en lugar de minúsculas
* El programa en donde más defectos se han removido, es el programa 4, ya que presento muchos errores de sintaxis.
* No todos los errores son de sintaxis, también eh tenido errores acerca de funciones, esto es:
  + No se encuentra la función que se llama
  + No es el nombre de la función llamada
  + Los valores que devuelven en la función no corresponden
  + Los valores enviados a la función no corresponden
* Se nota que no pasa de 50 Loc’s y la media está entre 20 y 30 Locs’ por lo que no está mal considerando que el código máximo ha tenido menos de 200 loc’s.

## DEFECTOS REMOVIDOS EN PRUEBAS POR PROGRAMA

La segunda fase en donde se encuentran mas defectos, es en las pruebas, podemos ver desde el programa 2, hasta el 6 cuales fueron los defects/KLoc removidos por programa.

Las pruebas que se utilizan para los problemas, casi nunca se tiene que buscar ya que el mismo libreo de PSP, te dice cuáles son las pruebas que debes de pasar, los problemas más encontrados en esta fase, se refieren a:

* El simple hecho de no planear y diseñar las pruebas, y hacer un diseño enfocado a resolver las pruebas, hace que se encuentren por lo menos 15 defects/loc por programa.
* En todos los programas me falto diseñar o tomar en cuenta las pruebas, y causa que no existiría ningún programa sin defectos en esta fase, las únicas pruebas que empecé a diseñar fueron las del programa 6, y aun así tuve que corregir defectos.
* La mayor parte de los defectos que encuentro en esta fase, es que la lógica no corresponde a los resultados de las pruebas, esto se debe a que no se diseña bien, o no se planea cierto funcionamiento en el programa.
* Otro tipo de defectos que eh encontrado son, los defectos de función, datos que mando mal a las funciones, direcciones erróneas o matrices mal señaladas.
* Los defectos lógicos deben de prevenirse en la parte de diseño y planeación, ya que, si se llega a las pruebas y se encuentran defectos de esas fases, el proceso se tiene que regresar demasiado, ya sea dos o tres fases atrás, en donde se supone que se debieron de haber previsto.
* Aunque la fase de pruebas debe de ser critica, aun después de esta fase eh encontrado defectos de los programas en la fase de postmorten o después de liberar el programa.

## DEFECTOS INYECTADOS A LA FECHA POR FASE

Durante el curso se han realizado 6 programas, dentro de los cuales 5 se han realizado con los procesos del PSP.

De las dos fases en las cuales se inyectan un total del 80% defectos son, codificación y diseño. Analizando en la gráfica podemos ver que:

* En la fase que tenemos más del 50% de defectos son en codificación
* La fase de diseño presenta menos defectos que la codificación
* El programa 2, se nota que la cantidad de defectos por código y la cantidad de defectos por diseño representan una similitud, ya que en ambos casos los defectos generan una punta en la grafica
* Con diferencia a el programa 5, donde fue el primer programa que empecé a tomar en cuenta los procesos de diseño del PSP, todos los demás presentan una variante de 20 puntos en la gráfica de defectos.
* También algo que ayudo mucho a que se cometieran tantos defectos en las fases de diseño y codificación es una buena estimación.
* Los procesos de estimación de loc’s ayudaron a reducir una gran parte del diseño al 40 %.

## DEFECTOS REMOVIDOS A LA FECHA POR FASE

Por otra parte, tenemos las fases en las cuales se remueven defectos, estas fases del proceso del PSP que comprendemos son 4:

* Test-pruebas
* Comp- compilación
* CDR
* Des.Rev

De estas fases solamente realizamos concretamente dos, test y Comp, las otras dos fases o procesos, no los utilizamos ya que el nivel del PSP que tenemos, no lo requiere.

De estas dos fases denotamos que:

* En este caso si tenemos datos para el programa 1, y se remueven casi 34 más en test que en compilación, esto debe de ser a que el programa representaba mayor cantidad de diseño.
* En cuanto a los demás problemas, notamos que tres de los seis programas llegan a 100 puntos en test, estos son el 2,3 y 5, pero en la parte de compilación no presentan la misma igualdad, ya que el programa 2 tiene apenas 22 y el 3 tiene 60, en cuanto al 5 tiene solamente 30. Es mucho lo que varía, por lo que no se puede decir que cuando uno es constante el otro no es igual, siempre se generan distintos errores y se remueven en diferente fase.
* Para los programas 2,3 y 4 la forma de planear y diseñar el programa fue la misma, con pocas variantes, y al final se ve que en test si se llegan a bajar los defectos removidos, esta misma característica se ve en los programas 5 y 6 que se hicieron con otros procesos distintos a los programas anteriores.
* Lo que aún no tengo un control en el proceso es en compilación, ya que muchos de los defectos que se remueven ahí, son de sintaxis o lógica, pero no sé cómo controlar o evitar hacer tantos de ellos, las gráficas presentan datos que no son concretos, no se tienen alguna línea que seguir o se nota alguna mejora, simplemente parecen datos random.
* Los puntos van de 20 a 60 el más alto en cuanto a defectos removidos en la fase de compilación.

# ANALISIS DE DEFECTOS MEDIANTE PARETO

El análisis de datos mediante Pareto, es una de las maneras más fáciles de solucionar los defectos que cometemos, el porqué de nuestros errores y en que parte la cometemos nos los dice el método de Pareto, el por qué los cometemos, es un tema individual.

El método de Pareto, corresponde a clasificar los errores en una determinada forma, este es asignándole un número a el tipo de errores, estos están clasificados así:

* 10-Documentacion
* 20-Sintaxis
* 30-Build, Package
* 40-Assignment
* 50-Interface
* 60-Checking
* 70-Data
* 80-Function
* 90-System
* 100-Enviroment

La clasificación de estos esta mejor descrita en la guía de PSP, dentro de los procesos describimos que, y cuantos errores cometemos y sobre qué tipo caen, de esa manera nosotros vemos cuales son los errores y el tipo que más se cometen y hacemos una forma de evitar hacerlos.

Además, el Pareto nos ayuda a calcular el tiempo en que se removieron los defectos y el tipo, esto nos ayuda para ver qué tiempo perdimos y porque se perdió.

El método nos presenta dos gráficas diferentes que analizaremos:

* Defectos removidos y fase de los defectos
* Tiempo de corrección de defectos

## DEFECTOS REMOVIDOS Y FASE DE LOS DEFECTOS

La primera grafía que analizaremos es la de defectos removidos por tipo, mediante el analices del método por Pareto, la clasificación de los defectos va de 10 en 10 desde el 10 al 100.

Se escribieron 6 programas, consiguiendo un total de 47 defectos de los cuales podemos decir que:

* De los dos tipos de defectos que se cometieron más son del tipo 10 de documentación y el tipo 80 que es de función.
* Los tipos 10, fueron 15 defectos, estos fueron por fallas en la documentación tanto del proceso de PSP, como en la documentación del programa, en cuanto a documentación de del PSP me equivoque al hacer los templates de size estimated, time log, defect log, Summary, etc., esto debido a la falta de experiencia, o de uso, ya que o era la primera vez que los utilizaba o también, mi falta de preguntar qué era lo que tenía que hacer concretamente. Los defectos en la parte de la documentación del programa, cae más en documentar código, palabras mal escritas, diseños que no se entendían esos eran los dos principales que me pasaron.
* Los del tipo 80, eran por funciones que utilizaba mal mandaba a llamar de una forma errónea, esto es porque las funciones que quería utilizar, no las conozco bien, y es uno de los problemas que más tengo, el utilizar funciones que no domino al cien, y pretender antes de entender su funcionamiento, que me van a servir. Muchos errores fueron que las escribí mal, o me faltaba algún argumento para enviar, o hasta un argumento que recibía. El simple hecho de que sabía que existían, no era para que las quisiera utilizar a si sin más, pero lo hice y bueno no fue nada bueno.
* Muchos de los errores cometidos del tipo 80, también hizo que tuviera errores de tipo 40, que es la asignación, por no saber cómo funcionan las funciones. Y bueno muchos más, fueron detalles de planeación en cuanto al programa, variables que no sabía que debían de comenzar de una manera explícita o con un valor detallado. Asignaciones de simplemente hacían que no corriera el código o fallaran en las pruebas.
* 4 errores del tipo 20 que son relativamente pocos, pero se encuentran, esto se debe más a distracciones, y que no me doy cuenta que me faltan letras o “;” dentro del código.
* Errores del tipo 30, 50 y 70, por lo menos una vez los eh cometido, estos son más por qué. Del tipo 30, una función no se encontraba en el paquete o la librería que está utilizando. Del tipo 50 de Interface, los datos no se presentaban de forma correcta, ya que no eran correctos. Y los de 70 Data, son solamente por datos duros mal introducidos, datos que se debieron de poner como un número mayor a 0 y solamente se tomó en cuanta 0.
* Los errores que jamás eh cometido son los de tipo 90 y 100, ya que nunca me a fallado el sistema operativo o el ambiente de trabajo.

## TIEMPO DE CORRECCION DE DEFECTOS

En la segunda grafica que analizaremos es la del tiempo de corrección de defectos, como podemos ver, todo esto está en minutos.

El tiempo que se tardan en corregir los defectos, se guardan en el defect log, analizando la gráfica podemos notar que:

* Los errores lógicos nos cuestan mucho tiempo dentro del desarrollo de los problemas, esto debe de modificarse y hacer una mejor planeación y diseño, para reducir este tiempo sustancialmente, si cada programa está tardando aproximadamente 100 minutos y son 10 minutos que se pierden reparando defectos de diseño, estamos diciendo que 10% del tiempo dedicado a la producción del código lo gastamos solamente en remoción de defectos de lógica, sin contar todos los demás.
* Los defectos de documentación, son los segundos que más tiempo tardamos en reparar, esto es debido a que la localización y el cambio no se hacen tan tardados, pero son muchos los que se presentan.
* Errores de asignación con 11 minutos, son otro tipo de errores que a pesar de que no son tan difíciles de corregir, la búsqueda de ellos causa mucho tiempo perdido.
* Para errores como el 20, que es de sintaxis, no se tarda mucho tiempo en la corrección, pero si son muchos los defectos que se han echo de este tipo.
* Todos los demás errores se corrigen rápido, y lo único que quita más tiempo es la escritura y redacción de su remoción o de su especificación.
* Igual que en la gráfica anterior, los tipos de datos 90 y 100 no tienen ningunos tiempos, ya que jamás se han realizado.

# TABLAS DE ANALISIS DE DATOS

Analizaremos los datos acerca de las loc’s y los defectos hechos, en constante interacciona con su tiempo de desarrollo.

Podremos ver el tiempo y las líneas de código en cada proyecto y como afecta su manipulación de tiempos y defectos.

## DENSIDAD DE LOS DEFECTOS Y FASE DE LOS DEFECTOS

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Defect Densities** | | | | **Compile and Test Defects** | | | |
| Program Number | New and Changed LOC | Total Defects | Defects per KLOC | Defects found in compile | Compile defects per KLOC | Defects found in test | Test defects per KLOC |
| 1 | 0 | 6 | 0 | 1 | 0 | 2 | 0 |
| 2 | 26 | 5 | 192 | 1 | 38 | 4 | 154 |
| 3 | 133 | 5 | 38 | 3 | 23 | 2 | 15 |
| 4 | 133 | 14 | 105 | 7 | 53 | 2 | 15 |
| 5 | 90 | 6 | 67 | 2 | 22 | 4 | 44 |
| 6 | 106 | 5 | 47 | 3 | 28 | 1 | 9 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Totals | 488 | 41 | 84 | 17 | 35 | 15 | 31 |

Los defectos encontrados por loc’s, son cambiantes, debido a la complejidad de los programas, mas sin en cambio representan datos coherentes de cómo y cuándo el desarrollador puede cometerlos.

Para la densidad de defectos notamos que:

* Los defectos encontrados no siempre delimitan la misma cantidad de loc’s erróneas, esto quiere decir, que hay defectos que valen poco y son muchos a defectos que valen mucho y son pocos.
* Para la mayor cantidad de defectos encontrados es el programa 4 con 24 defectos y una densidad de loc’s de 105, con la mayor densidad de defectos.
* Hasta el momento por cada 84 líneas de código cometo un error, esto es un buen limite, pero siento que se puede mejorar.

Pala la diferencia entre compilación y pruebas, podemos observar que:

* La mayor cantidad con 17 defectos encontrados es en compilación.
* Los 15 errores encontrados en test corresponden a que, en cada 31 líneas de código, se encontraran un defecto en test.
* Y por cada 35 líneas de código se encontrará un error en compilación.
* Lo que lleva a pensar que los errores en compilación no quitan tanto tiempo o importan tanto, como los errores que son encontrados en el test.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Number Injected** | | **Percentage Injected** | | **Number Removed** | | **Percentage Removed** | |
|
| Type | Design | Code | Design | Code | Compile | Test | Compile | Test |
| 10 | 0 | 1 | 0% | 2% | 0 | 0 | 0% | 0% |
| 20 | 0 | 4 | 0% | 9% | 4 | 0 | 9% | 0% |
| 30 | 0 | 1 | 0% | 2% | 0 | 1 | 0% | 2% |
| 40 | 0 | 8 | 0% | 17% | 7 | 1 | 15% | 2% |
| 50 | 0 | 1 | 0% | 2% | 1 | 0 | 2% | 0% |
| 60 | 1 | 0 | 2% | 0% | 0 | 2 | 0% | 4% |
| 70 | 0 | 1 | 0% | 2% | 0 | 1 | 0% | 2% |
| 80 | 4 | 10 | 9% | 21% | 5 | 10 | 11% | 21% |
| 90 | 0 | 0 | 0% | 0% | 0 | 0 | 0% | 0% |
| 100 | 0 | 0 | 0% | 0% | 0 | 0 | 0% | 0% |
| Total | 0 | 0 | 0% | 0% | 0 | 0 | 0% | 0% |

Observemos los tipos de errores cometidos, así como sus datos:

* Los porcentajes de datos erróneos inyectados son altos en algunos casos.
* En cuanto a los datos inyectados, se observa que hay más defectos del tipo 80 inyectados en código, lo que nos indica que no se hace un buen diseño, ya que son defectos de lógica.
* También hay muchos datos de tipo 40 inyectados en código el 17% de todos, estos errores son igual de lógica, se deberían de evitar en esta fase.
* Tenemos más defectos removidos en compilación, que no llegan ni a test, la mayor cantidad de ellos son de asignación, está fallando el diseño y no estoy escribiendo bien el código.
* El porcentaje más alto es de 21% en renovación de defectos en test aun del tipo lógico 80, tengo más defectos de análisis y diseño que de sintaxis y redacción de código.

|  |  |  |  |
| --- | --- | --- | --- |
| **Defect Type** | **Number of defects through Compile** | **Number of defects found in Compile** | **Percentage of Type found by the Compiler** |
| 10 | 5 | 0 | 23% |
| 20 | 0 | 4 | 18% |
| 30 | 0 | 0 | 0% |
| 40 | 0 | 7 | 32% |
| 50 | 0 | 1 | 5% |
| 60 | 0 | 0 | 0% |
| 70 | 0 | 0 | 0% |
| 80 | 0 | 5 | 23% |
| 90 | 0 | 0 | 0% |
| 100 | 0 | 0 | 0% |
| Total | 5 | 17 | 22% |

Parte importante del proceso para encontrar defectos es la compilación, ella podemos encontrar defectos en nuestro diseño y código de una manera fácil.

Analizando la tabla encontramos que:

* Son más los defectos que se encuentran en compilación que los que se inyectan.
* La mayoría de defectos hechos en compilación, fue por la reparación de defectos encontrados en la misma fase.
* Los defectos más encontrados siguen siendo los de asignación y lógica, hace falta más diseño.
* Errores que también se encuentran en compilación son de sintaxis, pero son muy pocos.
* La compilación ayuda a eliminar una gran cantidad de defectos, más aún así muchos de ellos pasan a la fase de pruebas
* Se deben de eliminar más defectos antes de que introduzcan.

## TIEMPO DE CORRECCION DE DEFECTOS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Defect Fix Times | | | | |
|  |  | Defects found in compiling | Defects found in testing | Total defects found |
| Defects | Tot. fix time | 5.00 | 10.00 | 15.00 |
| injected in | Tot. defects | 1.00 | 4.00 | 5.00 |
| designing | Avg. fix time | 5.00 | 2.50 | 3.00 |
| Defects | Tot. fix time | 18.00 | 40.00 | 58.00 |
| injected in | Tot. defects | 16.00 | 9.00 | 25.00 |
| coding | Avg. fix time | 1.13 | 4.44 | 2.32 |
| Total | Tot. fix time | 23.00 | 50.00 | 73.00 |
| defects | Tot. defects | 17.00 | 13.00 | 30.00 |
| injected | Avg. fix time | 6.13 | 6.94 | 5.32 |

Solamente para las dos fases, compilación y pruebas. Tenemos un total de 30 defectos de los 47 total.

La tabla nos indica que:

* El tiempo promedio de renovación de defectos es de 5.32 defectos por cada 30 defectos corregidos en 73 minutos.
* Tardo más tiempo corrigiendo defectos de lógica en las dos fases.
* Hay menos defectos encontrados en test, pero es más tardado corregir esos defectos, por cada uno tardo casi 7 minutos.
* De los defectos encontrados en compilación tardo 6 minutos en corregirlos, pero son muchos.
* Aproximadamente inyecto 25 defectos en codificación de los 30 totales, lo que indica que no se está haciendo bien el diseño y la planeación ya que todo se hace en codificación.
* Los únicos 5 defectos de diseño, que tardaron 15 minutos en resolverlos, lo que nos indica que son 3 minutos mínimo por cada uno de ellos.

# ANALISIS DE ESTIMACIONES

Para hacer un buen proyecto debe de haber una buena estimación, de lo contrario, todo puede salir mal. La gran parte del desarrollo del software se enfoca, en qué y cómo se debe de hacer los procesos y al mismo tiempo, calcular el tiempo de todos ellos.

Las estimaciones se hacen en tiempo y dos métricas más que nos ayudan a calcular cuánto nos tardaremos son:

* LOC’s
* Productividad

Sabiendo cuanto tardaremos en tamaño y cuanto nos tardamos haciendo determinados tamaños, nuestra planeación de tiempo, llega a ser muy exacta.

## ESTIMACIONES DE TAMAÑO

Empezare por analizar la planeación en loc’s de los programas, teniendo en cuanta que fue lo que planee contra lo que fue en realidad.

* Para los programas que he realizado el tamaño actual no rebasa las 133 líneas de código, teniendo como máximos a los programas 3 y 4.
* Del programa 1 no se tiene datos, porque no se utilizó ningún método.
* La mayoría de los códigos varían entre 20 líneas de código.

Las estimaciones tratan de ser mejores a lo largo del proceso del PSP. Mas no siempre se consigue.

* Al principio de mis estimaciones, me faltaban líneas de código por planear.
* El error muestra que, aunque ahora sobre estimo de mas, se tiene que acercar a 0.
* Las loc’s del programa 5 se sobre estimaron en gran parte.
* Para el programa 6 la mejoría es notoria ya que, el error se va acercando más y más a 0.

## ESTIMACIONES DE TIEMPO

El tiempo de desarrollo de los programas depende de todas las fases del proceso, por lo que tiende a ser alto, aunque el programa sea fácil.

La grafica nos muestra que:

* Coherentemente los desarrollos van de 1 a 4 horas de desarrollo.
* Los programas en los cuales me eh encontrado con más problemas de lógica son el 4 y el 5.
* El problema más fácil en cuanto a complejidad fue el 3.

Como podemos observar mejor, el error en cuanto a tiempo estimado en este caso es más bajo.

* Esto se debe a que se realiza mejor la estimación de tiempo y no de línea.
* O que el tiempo estimado se está sobre pasando, pero al final termina siendo el indicado.
* La falla del error corresponde a saber qué es lo que se va a hacer y planearlo, con seguridad.
* La herramienta nos ayuda a planear, pero si no se utiliza bien, todo va a salir muy diferido.
* Tuve problemas con la planeación ya que no entendía como utilizar el size estimate.

Como se logra ver, el tiempo planeado para los códigos, se sobre estima al final. Pero los defectos encontrados hacen que sea parecido a lo que realmente es.

Bien si tomamos en cuenta todos los tiempos.

* El tiempo de desarrollo, es poco comparado con el tiempo del proceso.
* Me tardo más tiempo corrigiendo el trabajo que en planear y diseñar.
* Cuando mejor planeo el proceso pinta bien.
* Si implemento algo nuevo para diseñar no se nota constantemente en el tiempo de desarrollo, ya que sigo teniendo muchos problemas, y tardo el mismo tiempo en corregirlos.

A lo largo del tiempo que llevamos cursando la materia. Se implementan nuevos procesos, así como nuevas platillas para el proceso.

Pero como vemos en la gráfica no siempre se tarda mucho tiempo aprendiendo los procesos, notemos que:

* Para el programa 2 cuando se empezó a implementar el PSP, el tiempo de registro en postmortem era muy alto.
* Cada vez que se trabaja en PSP se disminuye el tiempo en esta fase.
* Los errores que se cometen son más del programa que del proceso.
* Solamente cuando se cambia de nivel del PSP es cuando hay un pequeño tiempo de aprendizaje y después vuelve a bajar.

Los tiempos de compilación, nos dan a conocer cuántos defectos tenemos y tardamos en corregirlos.

* Varia muchos el tiempo porque los defectos en cada programa son diferentes.
* Cuando más tiempo existe en compilación es porque las funciones que invocaba no funcionaban como quería o eran erróneas.
* La sintaxis mala afecta el tiempo de compilación.

Los tiempos de pruebas, cambian constantemente, podemos observar que:

* Para los programas 2 y 3 los tiempos bajaron en buena forma, llegando casi a 2.
* Para el programa 4 y 5 cuando se implementó un nuevo nivel del PSP, los tiempos se elevaron.
* para el programa 6 el tiempo de pruebas bajo en considerable forma, casi llegando a nuestro mejor tiempo que fue el del programa 3 con 5.5.
* los tiempos de pruebas deberían de estar más bajos, ya que casi es el mismo tiempo de diseño.
* Cuando trato de diseñar, se suma más tiempo a las pruebas, en lugar de disminuir.
* Para el programa 6 las pruebas de planearon y se diseñaron de una buena forma.

Los tiempos de compilación y pruebas, son demasiados a lo largo de todos los programas.

Se está planeando más tiempo para corregir defectos que para hacer o diseñar, esto debe de cambiar y tratar de bajar los tiempos.

Un 30% de los tiempos estimados, se gastan en corregir errores.

# TABLA DE PRODUCTIVIDAD POR PROGRAMA

La productividad está enfocada a cómo hacemos los trabajos y para que los hacemos.

En cuanto a las loc’s que se hacen por hora podemos decir que bueno hasta el momento las líneas de código realizadas equivalen a el tiempo de los programas hechos.

No tenemos nada más con que comparar la productividad, hasta que el nivel de PSP nos lo proponga. Solamente podemos tener en cuenta que sin duda la productividad va mejorando cuando más concreto se sabe el proceso del PSP. Si se comenten errores que nos lleven a cambiar nuestras estimaciones o hacer más tiempos para el programa, nuestra productividad más va a bajar.

Un gran punto de vista que puedo agregar de porque la productividad en los primeros programas es más alta y luego baja y después se incrementan, es debido a que ya había trabajado antes con el proceso de PSP, pero solamente hasta el programa 3, por lo que ya sabía más o menos, cuantas líneas de código me iba a hacer y el tiempo que tardaría. Esto sin pensar en que los programas fueron diferentes a cierta manera, por eso causo que mi productividad fuera muy alta y fuera bajando hacia los programas que eran nuevos, así como los procesos.

# DESEMPEÑO PERSONAL AL USAR PSP

Bueno para esta parte en especifico, queda mencionar que, el proceso del PSP mejora no solamente actiudes de trabajo, si no tambien personales.

Al estar llevando esta materia, me eh dado cuanta que mi vida cambia a ser una vista de proceso, enfocar todo a PSP, esto es medir tiempos, planear actividades y diseñar lo que voy hacer en mi vida.

En cuanto a la productividad del desarrollo del software puedo mencionar que:

* Planear es la parte fundamental del desarrollo, a una buena planeacion, un excelente proyecto.
* Cabe mas la experiencia que la inteligencia, si se sabe como hacerce las cosas, se van a hacer bien.
* Si no se sabe como hacerlo, simpre hay alguine que te puede ayudar a solucionar tu problema.
* La codificacion debe de hacerce, simpre sabiendo y teneiendo en cuanta todo lo que se quiere hacer.
* Para saber como hacer proyectos, uno debe de conocerce completamente.
* Comociendo tu forma de trabajar, puedes planear tu proyecto con facilidad.
* Los errores mas que dañarte, te ayudan a ser mejor.
* El tiempo es fundamental en el desarrollo del software, pero la productividad del desarrollador puede salvar cualquier mala planeacion.
* Cuando sabemos como planear el tiempo de codificacion es menos y las pruebas son mejores.
* Cuando se sabe diseñar el trabajo es mas sencillo.
* Cuando las pruebas son realizadas correctamente y el tiempo es utilizado de manera correcta, ofreces un producto de calidad.
* El proceso de PSP, te ayuda a mejorar constantemente.
* Los datos recopilados durante un proyecto no solamente te ayudan a mejorar al instante, si no que te pueden ayudar en muchas cosas mas.

Sin duda alguna el curso conlleva muchas enseñanzas que favorecen el desempeño natural del desarrollo del software.

# RECOPILACION DE LOS PIP´S A LO LARGO DEL PROCESO

|  |  |  |
| --- | --- | --- |
| **Programa 2 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 1 | No tenía todos los documentos y materiales que se necesitaban a la mano. | Preparar todos los materiales antes de empezar a trabajar. |
| 2 | Falto hacer mejor el diseño y los requerimientos | Tener unos templetes de requerimientos y de diseño |
| 3 | Todo el trabajo se hizo bajo presión por poco tiempo | Planear la creación del proyecto con anterioridad |
| **Programa 3 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 4 | Muchas distracciones por el ambiente de trabajo | Trabajar en un lugar donde haya menos ruido |
| 5 | Falta añadir un mejor diseño | Utilizar el diagrama de clases, diagrama de casos de uso, diagrama de actividades y diagrama de estados. |
| **Programa 4 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 6 | Todo el trabajo se realizo bajo la presión del tiempo | Realizar el trabajo con anticipación |
| 7 | Hubo conceptos que no se entendían | Ir a las acesorias o enviar correo para preguntar sobre el proceso |
| **Programa 5 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 8 | No se tomo en cuanta el nivel del psp | Preguntar a el tutor por el nivel antes de empezar a realizar el programa |
| **Programa 6 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
|  |  |  |

# AREAS DE MAYOR PRIORIDAD PARA HACER MEJORAS EN LA SEGUNDA PARTE DEL CURSO

# CONCLUSIÓN

Para la mitad del aprendizaje de PSP. Me he encontrado con muchos aprendizajes, cosas buenas y malas al momento.

El simple echo de aprender el proceso de software, conociéndote tú mismo ayuda a mejorar. Las bases de PSP, nos enfocan a crear productos de calidad.

Aunque la gran parte de este curso, se enfocó en la planeación, me abrió los ojos a su importancia en todo tipo. El enseñarnos a planear usando los datos y a medir por medio de tiempos y líneas de código, es una gran ayuda para todos los tipos de proyectos.

Los datos enfocados a este proceso causan una mejora en mí, así que cometo mal y que es lo que debo de hacer mejor, por ejemplo.

La planeación es importante, el diseño es el núcleo del proyecto, la codificación sirve para hacerlo funcional y las pruebas para darle calidad.

Quiero seguir creciendo en mi proceso y sé que el PSP nos va a ayudar.