Análisis de datos sobre el desarrollo de software, orientado en un proceso definido PSP.

R5

Reporte final.

Juan Alberto Gutiérrez Canto

Mat.-24400063

**Datos Generales**

|  |  |
| --- | --- |
| **Materia** | Proceso Personal de Software 2 |
| **Tarea** | Reporte Final R5 |
| **Fecha de Liberación** | 19/07/2016 |
| **Fecha de Entrega** | 21/07/2016 |
| **Descripción General** | Ofrecer el conocimiento profundo del desempeño en el desarrollo de software actual y sus áreas de mayor prioridad para la mejora. |
| **Actividad** | Actualizar el proceso que utilizó para desarrollar el informe de mitad de período y utilizar este proceso de actualización para producir este informe final. |
| **Catedrático** | Jorge Rafael Aguilar Cisneros. |

**Alumno**

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

Tabla de contenido

[INTRODUCCION 4](#_Toc456886089)

[OBJETIVOS 4](#_Toc456886090)

[PROCESO DE ANALISIS DE DATOS DE PSP 4](#_Toc456886091)

[ANALISIS DE DEFECTOS 6](#_Toc456886092)

[TOTAL, DE DEFECTOS POR PROGRAMA 6](#_Toc456886093)

[DEFECTOS INYECTADOS EN DISEÑO 7](#_Toc456886094)

[DEFECTOS INYECTADOS EN CODIGO 9](#_Toc456886095)

[DEFECTOS REMOVIDOS EN REVICIONES DE DISEÑO 10](#_Toc456886096)

[DEFECTOS REMOVIDOS EN REVISIONES DE CODIGO 11](#_Toc456886097)

[DEFECTOS REMOVIDOS EN COMPILACION 13](#_Toc456886098)

[DEFECTOS REMOVIDOS EN PRUEBAS 14](#_Toc456886099)

[PORCENTAJE DE DEFECTOS INYECTADOS POR FASE 16](#_Toc456886100)

[PORCENTAJE DE DEFECTOS REMOVIDOS POR FASE 17](#_Toc456886101)

[DEFECTOS DE COMPILACION VS PRUEBAS 19](#_Toc456886102)

[ANALISIS DE DEFECTOS MEDIANTE EL METODO PARETO 20](#_Toc456886103)

[DEFECTOS REMOVIDOS POR TIPO 21](#_Toc456886104)

[TIEMPO DE DEFECTOS REPARADOS POR FASE 22](#_Toc456886105)

[TABLAS DE ANALISIS DE DATOS 24](#_Toc456886106)

[DENSIDAD DE LOS DEFECTOS Y FASE DE LOS DEFECTOS 24](#_Toc456886107)

[TABLA DE CORRECCION DE DEFECTOS 27](#_Toc456886108)

[ANALISIS DE ESTIMACIONES 28](#_Toc456886109)

[TAMAÑO ACTUAL 28](#_Toc456886110)

[ERROR DE LA ESTIMACION DE TAMAÑO 29](#_Toc456886111)

[TIEMPO DE DESARROLLO ACTUAL 30](#_Toc456886112)

[ERROR DEL TIEMPO ESTIMADO 31](#_Toc456886113)

[PORCENTAJE DEL TIEMPO EN PLANEACION 32](#_Toc456886114)

[PORCENTAJE DEL TIEMPO EN POSTMORTEM 33](#_Toc456886115)

[PORCENTAJE DEL TIEMPO EN POSTMORTEM + PLANEACION 34](#_Toc456886116)

[PORCENTAJE DEL TIEMPO DE COMPILACION 35](#_Toc456886117)

[PORCENTAJE DE TIEMPO EN PRUEBAS 36](#_Toc456886118)

[PORCENTAJE DE TIEMPO DE COMPILACION + PRUEBAS 37](#_Toc456886119)

[ANALISIS DEL PROCESO 38](#_Toc456886120)

[PRODUCTIVIDAD 38](#_Toc456886121)

[PRODUCTIVIDAD VS YEILD 40](#_Toc456886122)

[PRODUCTIVIDAD VS A/FR 41](#_Toc456886123)

[YEILD VS A/FR 42](#_Toc456886124)

[DEFECTOS DE PRUEBAS VS A/FR 43](#_Toc456886125)

[DEFECTOS DE PRUEBAS VS YEILD 43](#_Toc456886126)

[ANALISIS DE CALIDAD 44](#_Toc456886127)

[YEILD DEFECTOS REMOVIDOS 44](#_Toc456886128)

[PERDIDAD DEL COSTO DE CALIDAD 45](#_Toc456886129)

[APPRAISAL COSTO DE CALIDAD 46](#_Toc456886130)

[COSTO TOTAL DE LA CALIDAD 47](#_Toc456886131)

[APPRAISAL TO FAILURE RATIO 48](#_Toc456886132)

[REVISIONES DE LOC´s POR HORA 49](#_Toc456886133)

[REVISIONES DE LOC’s POR HORA VS YIELD-REVISIONES DE DISEÑO 50](#_Toc456886134)

[REVISONES DE LOC’s POR HORA VS YEILD- REVISIONES DE CODIGO 51](#_Toc456886135)

[REVISIONES DE LOC´s POR HORA VS YEILD-REVISIONES DE DISEÑO Y CODIGO 52](#_Toc456886136)

[DESEMPEÑO PERSONAL AL USAR PSP 53](#_Toc456886137)

[RECOPILACION DE LOS PIP´S A LO LARGO DEL PROCESO 54](#_Toc456886138)

[AREAS DE MAYOR PRIORIDAD PARA HACER MEJORAS CONTINUAS 56](#_Toc456886139)

[CONCLUSIÓN 58](#_Toc456886140)

# INTRODUCCION

El proceso personal de software PSP, nos otorga una manera de aprender a conocernos. La manera que utiliza el PSP es por medio de 3 niveles de procesos definidos, que ayudan a recabar datos específicos para el desarrollador y el proceso de desarrollo de software que ayuda a medir y alcanzar metas específicas para mejorar.

El curso en este caso nos tocó en dos partes, la primera que abarca del PSP0 al PSP1.1 y la segunda parte que fue del PSP1.1 al PSP2.1, en el cual nos tocó ver un proceso escalable, poco a poco se aprendió a hacer el proceso de PSP, y hoy en día tenemos definido que es lo que hacemos, como lo hacemos y de qué manera lo medimos.

El reporte final R5 presenta todo lo que se hizo durante estos dos cursos, cada ingeniero presenta una especificación concisa de su proceso, así mismo analiza la manera de mejorar.

# OBJETIVOS

Al momento que se trabaja con un proceso de PSP 2.1 se integran nuevas fases y métricas, las cuales nos ayudan a verificar la calidad de nuestro proceso de software, para este reporte, ponemos mayor enfoque a:

* Mejoras en las fases de desarrollo.
* Mejoras en el proceso PIP´s.
* Definición del mejor uso del proceso.
* Auto evaluación.

# PROCESO DE ANALISIS DE DATOS DE PSP

# 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 el 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 segunda parte del curso en PSP2 los defectos bajan y siguen bajando, ya que se incrementó el proceso y ayudo a la eliminación temprano de defectos.
* Del programa 8 podemos notar que es mayor el número de defectos por Loc´s que del 9, ya que aun que fueron solamente pocos defectos, el tamaño del código fue pequeño, por lo cual se llegan a los datos actuales.
* Al incrementar tareas en la fase de diseño, ayudo a mejorar.

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.

EL proceso de PSP2.1 ayudo a evitar errores en la mayor parte de las fases, esto hizo que se redujeran en un número muy grande y constante.

## DEFECTOS INYECTADOS EN DISEÑO

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.
* Para los programas del curso de PSP2 ya se contaba con diseño, y esto represento que creciera la cantidad de errores encontrados.
* Aunque se descubren más errores en esta fase, esto simplifica su reparación temprano y por lo tanto no se pierde mucho tiempo.
* A pesar de eso, con el paso del tiempo se mejoró y se dejaron de introducir tantos defectos como antes.

El diseño causo un gran impacto, ya que se descubrían antes los defectos y ayudaba a las otras fases a liberar tiempo de corrección de errores.

## DEFECTOS INYECTADOS EN CODIGO

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.
* En los programas 7,8 y 9 se tiene una disminución muy notable de los defectos loc’s.
* Se logró tener un programa fuera de errores en esta fase, que fue el 8, a pesar de que era muy fácil, el hecho de planear y administrar bien el programa causo que se lograra esto.

El proceso facilita la opción de evitar ingresar defectos en las fases del software, a pesar de que no se puede eliminar todos los defectos, se pueden evitar la gran parte.

## DEFECTOS REMOVIDOS EN REVICIONES DE DISEÑO

Esta es una nueva fase, introducida desde el proceso PSP2, consta de hacerle revisión a nuestro diseño, siguiendo determinados templetes y con ayuda de herramientas para checar los ciclos y funcionalidades básicas.

Como podemos observar en la gráfica:

* Los únicos datos que se tiene son desde el 7.
* Los defectos bajaron constantemente, como se utilizaba más la fase.
* En el programa 9 se llegó a tener casi nula la búsqueda de defectos.
* A pesar de que los defectos removidos en esta fase fueron disminuyendo, es porque ya no se inyectaban muchos defectos, no se refiere a que dejo de funcionar las revisiones.
* En el programa 7 se removió una gran cantidad de defectos en esta fase, y ayudo a entender que se debía de poner más atención a la hora de hacer el diseño de un programa.

La fase es de gran ayuda, ya que podemos evitar liberar errores que al final, nos costarían tiempo y esfuerzo poder eliminar en fases mas tardías.

## DEFECTOS REMOVIDOS EN REVISIONES DE CODIGO

Esta fase también es nueva ya que solamente la empleamos desde el proceso PSP2.

La fase consta en diferentes templetes para la revisión de código, esto se centra más en la sintaxis y buen uso de las palabras claves del lenguaje de programación, es empleada desde el proceso de PSP2 y solamente es visible para los programas 7,8 y 9.

Podemos observar de la gráfica que:

* Tiene un modo a disminuir los defectos removidos en la fase, debido a que ya no existían muchos inyectados.
* En el programa 8 no se removió ninguno, ya que no existían defectos inyectados, así que a no haber defectos no se pueden eliminar.
* La cantidad de líneas de remoción de defectos corresponden al tamaño de los programas.
* A pesar de que los programas sean grandes como el 9, podemos observar que la cantidad de defect´s loc´s es menor a la del 7, que contiene un poco más de la mitad de tamaño que el 9.

Para el programa 10 no se tiene registros ya que no se tomará en cuenta para el reporte final del curso.

## DEFECTOS REMOVIDOS EN COMPILACION

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.
* Una vez que se implementó el proceso PSP2, la cantidad de loc’s de defectos que se removieron bajaron a 10 loc´s.
* La utilización de diseño, ayuda a que se tenga buena relación con lo que se tiene que codificar.
* Las revisiones de código ayudan a bajar los defectos sustancialmente.
* De los últimos 3 programas que son 7,8 y 9, se observa que la cantidad de remoción es de menos de 10 loc´s.
* El programa 8 no se removieron defectos en compilación, ya que no había defectos existentes.
* La eliminación de defectos bajo el 80 % al final del curso.

Que se eliminen menor cantidad de defectos en esta fase, significa que la sintaxis a mejorado y el diseño es más conciso, se sabe lo que se hace y es más fácil escribir código.

## DEFECTOS REMOVIDOS EN PRUEBAS

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.
* Una vez que se empezó a implementar el diseño, desde el programa 7, mejoro el número de remoción de defectos.
* El diseño aseguro que los problemas que se encontraban en esta fase bajaran muchísimo.
* Las revisiones de diseño, nos ayudaron a filtrar los defectos y evitar que lleguen a fases más tardías.
* Para el programa 8, se llevó un proceso muy bueno, y al llegar a esta fase, no se tuvo ningún defecto.
* A pesar de que pasaron defectos a esta fase en los programas 7 y 9, su eliminación fue rápida.

Al llegar al final del curso los defectos que llegaron a esta fase fueron menos del 70% que, al principio, no se encontraron defectos después del postmorten y se aseguró un buen uso del sistema al final.

## PORCENTAJE DE DEFECTOS INYECTADOS 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 %.
* Para los últimos programas se observa que la inyección de defectos en código, bajo ligeramente, lo que nos dice, que seguimos inyectando muchos defectos en esta fase, mas sin en cambio, se está mejorando.
* Para el diseño, se tiene una mayor inyección de defectos, debido a que si se hizo un diseño formal para los últimos programas.
* Los defectos inyectados en diseño, fueron reduciendo conforme paso el tiempo.

Se tiene una baja de defectos inyectados en fases tardías del proceso de desarrollo de software.

## PORCENTAJE DE DEFECTOS REMOVIDOS 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.
* Para la segunda mitad del curso tenemos dos fases nuevas.
* Las revisiones de diseño lograron eliminar más del 70% de defectos existentes hasta su fase, lo que es muy bueno, ya que no pasaran a otras fases.
* Las revisiones de código presentan una eliminación del 80% o más, para su fase, se están eliminando muchos defectos antes de la compilación.
* Para la codificación, una vez se logró el 100% y así evitar que llegaran defectos a la fase final, aun así tiene el 90% de eliminación de defectos a su fase.
* En test, presentamos el 100% de defectos removidos, lo que nos llega a ver, que después del lanzamiento, no se presentaron errores y no se tuvo que corregir nada más.

Las nuevas fases de revisión, ayudan a que la compilación y pruebas, tengan pocos defectos para eliminar y mejore su porcentaje.

## DEFECTOS DE COMPILACION VS PRUEBAS

Una de las maneras en la que podemos entender mas sobre la remoción de nuestros defectos es en la comparación de los defectos de compilación y pruebas.

# ANALISIS DE DEFECTOS MEDIANTE EL METODO 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 POR TIPO

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 DEFECTOS REPARADOS POR FASE

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.
* A pesar de que los defectos de función ya se resuelven más rápido por el diseño y su búsqueda temprana, los datos del programa anterior nos orillan a que se muestre una no mejora.

# 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 | -79 | 6 | -76 | 1 | -13 | 2 | -25 |
| 2 | 105 | 5 | 48 | 1 | 10 | 4 | 38 |
| 3 | 133 | 5 | 38 | 3 | 23 | 2 | 15 |
| 4 | 133 | 14 | 105 | 7 | 53 | 2 | 15 |
| 5 | 90 | 7 | 78 | 2 | 22 | 4 | 44 |
| 6 | 106 | 4 | 38 | 3 | 28 | 1 | 9 |
| 7 | 213 | 21 | 99 | 0 | 0 | 2 | 9 |
| 8 | 73 | 4 | 55 | 0 | 0 | 0 | 0 |
| 9 | 320 | 8 | 25 | 1 | 3 | 1 | 3 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Totals | 1094 | 74 | 68 | 18 | 16 | 18 | 16 |

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.
* Después de terminar el curso por cada mil líneas de código hay 68 errores, eso es una respuesta favorable a la mejora del proceso.

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.
* El proceso en su mejora arroja que ahora son 16 defectos en compilación y pruebas, lo que se remueve por cada 1000 lineas de código.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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.

## TABLA 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.

## TAMAÑO ACTUAL

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.
* Como se observa, solamente es para ver el tamaño actual de los problemas, presentando su máximo en el programa 9.

## ERROR DE LA ESTIMACION DE TAMAÑO

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.
* La planeación más exacta que se tuvo fue la del programa 8.
* Para el programa 9 se tiene una deficiencia del proceso que iba a la baja, ya que se decidió hacer las funciones nuevamente, ya que las que se reutilizarían, no funcionaban para nada.

## TIEMPO DE DESARROLLO ACTUAL

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.
* En cuanto a los últimos programas se puede observar que la complejidad lanzo los tiempos muy altos.
* Los últimos programas tuvieron un diseño formal.
* Las revisiones hacen que el tiempo de los programas fuera más grande, pero se asegura la calidad.

## ERROR DEL TIEMPO ESTIMADO

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 cómo utilizar el size estimate.
* Suigue siendo un poco difícil como calcular los tiempos.
* La herramienta sirve para calcular los métodos, pero aún falta la experiencia para hacer una planeación de tiempo adecuada.
* Los requerimientos del programa no facilitaban la apreciación fácil de la estimación, se tenía que rebuscar un poco más y causaba tiempo perdido.

## PORCENTAJE DEL TIEMPO EN PLANEACION

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.
* Bajo el porcentaje de tiempo erróneo para la planeación.

## PORCENTAJE DEL TIEMPO EN POSTMORTEM

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.
* Una vez que se aprende y se utiliza más el proceso de PSP, causa hacerlo todo más rápido.
* Al asegurar la calidad del desarrollo, ahorro tiempo en esta fase.

## PORCENTAJE DEL TIEMPO EN POSTMORTEM + PLANEACION

El porcentaje sobre la planeación y el postmortem time, nos dicta que también se entiende el problema y se utiliza los métodos para la buena planeación y administración del proyecto.

Lo que observamos es que:

* En cuanto el proceso mostro más usos, los porcentajes bajaron.
* Para al final de los programas, el proceso se hizo familiar y la forma de aceparlo era más fácil, esto causo que el porcentaje bajara.
* Se aprendió a planear de una manera más plena y mejor.
* Las métricas mejoraron los tiempos.
* Los templetes ayudaron a que me diera cuanta que era lo que tenía que mejorar y los pip´s determinaron una solución franca.

## PORCENTAJE DEL TIEMPO DE COMPILACION

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.
* El hacer un mejor diseño y tener nuevas fases de revisiones, hicieron que los errores no llegaran a esta fase, por lo que su tiempo bajo más del 80% y en ocasiones llego a ser nulo.
* Los errores que llegaron a esta fase, al final de los programas, fueron errores de 1 minuto o menos tiempo, que son fáciles de resolver.

## PORCENTAJE DE TIEMPO EN PRUEBAS

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.
* Para el proceso PSP2, el tiempo de pruebas bajo, ya que no se tenían muchos errores que resolver y solamente se enfocaba en que la lógica funcionara bien.
* Las revisiones y los templetes de código, hacen una gran ayuda para que nuestro diseño sea correcto, y filtra los errores para no tener muchas complicaciones después.
* Se logró mejorar en los tiempos, a pesar de que los tamaños de los programas eran mas grandes.

## PORCENTAJE DE TIEMPO DE COMPILACION + PRUEBAS

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.
* Se bajó a menos del 15% de los proyectos aplicando el PSP2-PSP2.1
* Una de las finalidades del PSP es que no se tarde mucho tiempo en estas fases y se imprima más trabajo al principio para asegurar la calidad, lo que se logró ya que son pocos errores y muy poco tiempo el que se utiliza en la fase de pruebas y compilación.
* Los errores que llegan a esta fase después de la mitad del curso, son poco son fáciles de reparar, no tardamos mucho tiempo y se emplea mejor la calidad.

# ANALISIS DEL PROCESO

EL proceso de PSP, contiene muchas métricas para verificar que se está cumpliendo de una manera ideal y al mismo tiempo se mejora.

Dependiendo del nivel de PSP, podremos medirnos mejor y en diferentes áreas como son:

* Líneas de código
* Tiempo de desarrollo
* Productividad
* Yeild
* Remoción de defectos

Analizaremos estas métricas para entender más nuestro proceso, que es lo que se hace y como se puede mejorar.

## PRODUCTIVIDAD

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.

Una vez que se empleó un diseño mejor y se mostró una forma de que revisar y asegurarnos que era lo que se aria, la productividad se puede tomar como que bajo, ya que ahora se enfocaba mas no solamente a escribir código, si no a hacer un diseño de calidad, lo que cuesta más tiempo por línea de código, puedo asegurar que ahora mi productividad muestra no solamente lo que soy capaz de hacer si no que lo are con una buena calidad.

## PRODUCTIVIDAD VS YEILD

El yeild es una métrica para asegurar la remoción de defectos en una etapa temprana del desarrollo de software, esto es antes de la compilación.

Lo que podemos observar es que:

* Para los primeros programas no contábamos con esta métrica, ya que no se hacían revisiones de diseño y código.
* Las remociones después del programa 7 nos aseguran que se eliminara el 70% de los defectos antes de compilación.
* El yeild alcanzado a la fecha es muy bueno e indica que nuestra forma de trabajar tiene calidad.
* Para los proyectos más grandes es más difícil alcanzar un buen yeild, pero no imposible.
* Para problemas sencillos como el 8, se logró eliminar todo antes de compilación, lo que aseguro una calidad del producto del 100%.

## PRODUCTIVIDAD VS A/FR

Para determinar que también estás haciendo el proceso, nos encontramos con el appraisal to failure ratio, que es en pocas palabras como trabajamos nuestro proceso de acuerdo a nuestras fases.

Por lo que podemos observar que:

* Para los últimos problemas se tuvo una mejora en el proceso y se utilizó más tiempo en el diseño y codificación que en las pruebas y compilación.
* Estamos trabajando dos veces más en el diseño, lo que dice que se están pensando mejor como hacer las cosas.
* En el programa 8 que fue el que no llevo ningún defecto a la fase de compilación, tenemos un valor de casi 10, esto quiere decir que el diseño se enfocó también, que no solo permitió que las fases de pruebas y compilación fueran nulas, sino que también aseguro la calidad de manera plena.
* Se puede mejorar más aún si se siguen los procesos y templetes para el diseño.

## YEILD VS A/FR

Para esta comparación acerca de cómo es nuestra calidad de producto con cómo es la calidad de nuestro proceso, podemos ver que:

* El PSP nos ayuda a que no tengamos una gran cantidad de errores, siempre y cuando nos enfoquemos más en etapas tempranas.
* Si nosotros diseñamos y revisamos, así como codificamos de la mejor manera antes de la compilación, no solamente lograremos hacer más de lo normal, sino que la calidad de nuestro producto crecerá.

## DEFECTOS DE PRUEBAS VS A/FR

Podemos observar que mientras es mejor nuestro A/FR nuestro defect loc´s es muy poco, debido a que se imprime mayor cantidad de tiempo y esfuerzo a que no se tengan errores.

El caso en el cual es a/fr es mayor y el defect loc´s es grande es debido a la complejidad del proyecto y el tamaño en locs, ya que esto puede causar que tengamos más defectos, y estos se filtren hasta las pruebas.

Para los programas de un comienzo no podemos ver un a/fr claro, ya que aún no se contaban con fases como son revisión de diseño y código.

Al hacer diseño, podemos pensar en todo lo que el programa necesita y requiere, y si le agregamos las revisiones y un grado de atención mayor, obtenemos calidad.

## DEFECTOS DE PRUEBAS VS YEILD

Podemos observar que, si se tiene un proceso enfocado a la calidad, logramos disminuir la cantidad de defectos que llegan a fases complejas.

Para el programa 8 se alcanzó un yeild del 100% ya que no llego ningún defecto a la fase de pruebas, y como podemos observar, por lo mismo no se tiene ningún defect loc´s para el, esto es asegurar la calidad lo mejor posible.

# ANALISIS DE CALIDAD

## YEILD DEFECTOS REMOVIDOS

La métrica de yeild solamente la aplicamos desde el programa 7, por eso es que no tenemos ningún registro anterior a él.

Podemos notar que no se tiene un nivel constante del yeild, ya que la complejidad y el tamaño de los programas no era el mismo, y se puede tener variaciones constantes, pero a pesar de eso, logramos cumplir el objetivo del PSP, que nos decía alanzar un yeild de más del 70%.

El yeild más alto que alcance fue del 100%, eso fue a la dedicación del diseño y la implementación correcta de revisiones, además que el problema presentaba una manera fácil de resolverlo.

## PERDIDAD DEL COSTO DE CALIDAD

Cuando nosotros no trabajamos con calidad, nuestros productos, estamos constantes a que perdamos más tiempo y dinero del que teníamos predicho.

El costo de la calidad lo podemos definir de acuerdo a lo que haces o dejas de hacer. Para el PSP mientras mejor seas en el proceso, más calidad le dedicas a tus productos.

Durante los 9 programas, logramos ver como el costo de la calidad disminuye, haciendo que cada día, hagamos programas mejor diseñados y de calidad pura.

Si el costo de la calidad es alto, nos obliga a trabajar más en fases donde no se tendría que trabajar de más, o resolver problemas de otras fases.

## APPRAISAL COSTO DE CALIDAD

Podemos observar en esta grafica sobre el costo de calidad del appraisal, que si nosotros dejamos de tener defectos después de compilación, ahorraremos tiempo en buscar la solución de los defectos, ya que entre más tarde se localice el defectos, más tiempo lleva su reparación.

Para los programas anteriores, todo el problema los resolvíamos en las fases de compilación y pruebas, por lo que el appraisal es nulo.

## COSTO TOTAL DE LA CALIDAD

Podemos notar que la calidad aumenta conforme los problemas van pasando, al principio no se podía definir una buena calidad, pero con el tiempo, podemos definir que los programas tienen calidad o cero defectos para su lanzamiento, algunos antes aun de la compilación.

## APPRAISAL TO FAILURE RATIO

Podemos observar que realmente obtuvimos una muy buena ayuda por parte de los templetes de diseño y las revisiones, ya que antes de que yo pusiera empeño en hacer esto, todos los defectos se iban a la parte de compilación y pruebas.

Si teníamos todos los defectos en las últimas fases, el retroceso para ver que realmente era el error, y su corrección no solamente en la fase si no que en toda la documentación, era cansado y pesado, ahora con el nuevo proceso eliminamos defectos antes de llegar a compilación y los que llegan son muy fáciles de eliminar.

## REVISIONES DE LOC´s POR HORA

Podemos ver que las revisiones se encuentran arriba de las 200 loc´s por hora, lo que es recomendable según el autor del libro.

Pero no siempre se presenta esto ya que para algunas ocasiones se necita más revisión de diseño que de código, y en otras es deferente, que es lo que hace que esto sea cambiante, bueno pues se tiene que tomar en cuenta las forma como se revisó y la complejidad del problema, que puede ser de muy fácil a más complejo.

Para los demás programas no se tienen registro solamente los que llevaron un nivel de PSP2 o mayor.

## REVISIONES DE LOC’s POR HORA VS YIELD-REVISIONES DE DISEÑO

Para la fase de diseño, podemos observar que su revisión de acuerdo a las loc´s por hora:

* Entre menos líneas se revisen por hora es mejor, ya que cuando se analizaron de 600 a 800 se tuvo un yeild del 50%
* El punto que mejor resulto en mi caso, fue cuando se revisaron entre 200 y 400 por hora, con un yeild del 77%
* Aun estando debajo de las 200 loc´s por hora se obtuvo casi el 70% de yeild, que aún es bajo.

## REVISONES DE LOC’s POR HORA VS YEILD- REVISIONES DE CODIGO

Cuando se traga de hacer revisiones de código, el tiempo cambia, de acuerdo a las revisiones de diseño, ya que:

* Las loc´s mínimas para alcanzar un buen yeild es de 400 por hora.
* El mejor tiempo es de 500 loc´s por hora ya que fue en donde logre alcanzar un yeild del 65%, bajo aun para el 70% que se esperaría.
* De menos de 400 loc´s por hora, no se pueden encontrar errores claros, y el yeild es de 0%.

## REVISIONES DE LOC´s POR HORA VS YEILD-REVISIONES DE DISEÑO Y CODIGO

Si se analizan las dos fases de revisiones podemos llegar a ver que, con 100 loc´s por hora s alcanzo el mejor yeild que es de 100%, pero recordemos que fue un programa muy sencillo, así que lo mejor seria tomar un rango de 200 a 300 loc´s por hora para obtener un yeild de más del 70%, asegurando así la calidad de los proyectos.

# 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, siempre 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.
* Las revisiones no solamente se deben de hacer al final del proceso, si no constantemente.
* El diseño nutre el entendimiento y el producto desde adentro.
* El conocerse y saber en que mejorar, te ayudaran para toda la vida.

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** |
|  |  |  |
| **Programa 7 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 9 | No se entendio complementamente el problema a resolver | Tratar de poner mas atención y definir mejor el problema |
| 10 | Especificar mejor el diseño | Utilizar mejores procesos de PSP |
| **Programa 8 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 11 | No se sigue el calendario de planeación para el trabajo | Al momento de planear los días que se trabajaran, tener en cuenta las actividades fuera de la clase. |
| **Programa 9 A** | | |
| **No. Del PIP** | **Descripción del Problema** | **Descripción de la Propuesta** |
| 12 | No se entienden las funciones anteriormente echas | Hacer funciones para el reusó. |

# AREAS DE MAYOR PRIORIDAD PARA HACER MEJORAS CONTINUAS

# CONCLUSIÓN

El PSP es un proceso que te ayuda a conocerte no solamente en el ámbito enfocado al software, también de un modo muy personal.

La producción del PSP, nos da un enfoque a dar y hacer productos y procesos de calidad, cuidar mediante métricas y bases, nuestros productos, tratar de hacer un proceso de desarrollo más relajado, en el cual no solamente nos presionemos por la actividad que sigue o si se entregó a tiempo, hasta el saber si vamos bien o no dentro de un proyecto. Va más a ya, está enfocado a saber qué es lo que podemos y cuando lo podemos hacer, si es bueno hacerlo de alguna forma en específico o con algunos templetes utilizados anteriormente.

Este proceso ayuda a conocerte como desarrollador de software, y poder identificar como lo son los demás.

Además de que se enfoca en que se implementen procesos dentro del desarrollo y no solamente tratarlo de hacer de una manera ingenieril o a la hay se va. Te da herramientas para mejorar tus planeaciones como el size estimated, o aquellas que sirven para el diseño funcional, operativo o de estados. Y no se diga de la calidad de remoción y prevención de defectos, que te obliga a no gastar tiempo en fases tardías y arreglarlos en las fases donde se inyectaron.

Después del curso, me eh llegado a conocer y a mejorar, y sin duda alguna, el proceso de auto mejora, lo seguiré empleando.